



UNIVERSIDAD
DE MÁLAGA



ESCUELA DE INGENIERÍAS INDUSTRIALES

Ingeniería de Sistemas y Automática

TRABAJO FIN DE GRADO

CONTROL DE MOVIMIENTO DE UN INSTRUMENTO QUIRÚRGICO Y DISEÑO DE INTERFAZ GRÁFICA PARA LA CONFIGURACIÓN DE PARAMETROS Y COMANDOS DE MOVIMIENTO

Grado en

INGENIERÍA ELECTRÓNICA, ROBÓTICA Y MECATRÓNICA

Autor: **Pablo Fuentes Requena**

Tutor: **Carlos J. Pérez del Pulgar Mancebo**

Cotutora: **Irene Rivas Blanco**

MÁLAGA, julio de 2020



UNIVERSIDAD
DE MÁLAGA



**DECLARACIÓN DE ORIGINALIDAD DEL
PROYECTO/TRABAJO FIN DE GRADO**

D./ Dña.: Pablo Fuentes Requena.

DNI/Pasaporte: 79042040-X. Correo electrónico: pablofr918@gmail.com

Titulación: Ingeniería Electrónica, Robótica y Mecatrónica.

Título del Proyecto/Trabajo: Control de movimiento de un instrumento quirúrgico y diseño de interfaz gráfica para la configuración de parámetros y comandos de movimiento.

DECLARA BAJO SU RESPONSABILIDAD

Ser autor/a del texto entregado y que no ha sido presentado con anterioridad, ni total ni parcialmente, para superar materias previamente cursadas en esta u otras titulaciones de la Universidad de Málaga o cualquier otra institución de educación superior u otro tipo de fin. Así mismo, declara no haber trasgredido ninguna norma universitaria con respecto al plagio ni a las leyes establecidas que protegen la propiedad intelectual, así como que las fuentes utilizadas han sido citadas adecuadamente.

En Málaga, a 30 de julio de 2020

Fdo.:

RESUMEN

Este trabajo de fin de grado está enmarcado dentro de la robótica quirúrgica, y tiene como objetivo desarrollar e implementar un control de movimientos de un brazo robótico para el manejo de un instrumento laparoscópico. Para ello, habrá que tener en cuenta las restricciones de movimientos que impone el punto de entrada de la herramienta en el abdomen del paciente, conocido como punto de fulcro.

La implementación del control desarrollado se realizará en el entorno ROS (Robotic Operative System), utilizando PHYTOM como lenguaje de programación. De este modo, el programa implementado deberá comunicarse con el robot tanto para recibir información acerca de su localización actual, como para enviarle un nuevo comando de movimiento. Además, se diseñará e implementará una interfaz gráfica que permitirá tanto introducir un conjunto de parámetros para la ejecución del programa, como comandar al robot un determinado movimiento deseado.

Palabras clave: ROS, punto de fulcro, Python, robótica quirúrgica, control de movimientos.

ABSTRACT

This final degree project is framed within surgical robotics and aims to develop and implement motion control of a robotic arm to operate a laparoscopic instrument. For this, it will be necessary to take into account the movement restrictions imposed by the entry point of the tool in the patient's abdomen, known as the fulcrum point.

The implementation of the developed control will be carried out in the ROS (Robotic Operative System) environment, using PYTHON as the programming language. In this way, the implemented program must communicate with the robot both to receive information about its current location and to send it a new movement command. In addition, a graphical interface will be designed and implemented that will allow both the introduction of a set of parameters for the execution of the program, and the command to the robot of a certain desire movement.

Key words: ROS, fulcrum point, Python, surgical robotics, motion control.

ÍNDICE

1 INTRODUCCIÓN	5
1.1 Introducción a la robótica médica	5
1.2 Objetivos del TFG	9
1.3 Estructura del TFG.....	10
2 CONTROL DE MOVIMIENTO ALREDEDOR DE UN PUNTO DE FULCRO	11
2.1 Esquema de control	11
2.1 Representación de la posición	12
2.2 Representación de la orientación	12
2.2.1 Matriz de rotación.....	12
2.2.2 Ángulo-eje y Vector de rotación.....	13
2.2.3 Ángulos de Euler.....	15
2.3 Conversión entre las distintas representaciones de la orientación.....	15
2.4 Estimación del punto de fulcro	18
2.5 Movimiento alrededor del punto de fulcro	20
3 IMPLEMENTACIÓN.....	25
3.1 Introducción a ROS	25
3.2 Partes de ROS.....	25
3.2.1 Master	26
3.2.2 Topics	26
3.2.3 Nodos	28
3.3 URXdriver	34
3.4 URSim.....	35
3.5 UR3 de Universal Robots	35
4 EXPERIMENTACIÓN Y RESULTADOS	37
5 CONCLUSIONES Y FUTUROS TRABAJOS	47
6 BIBLIOGRAFÍA.....	49

1 INTRODUCCIÓN

1.1 Introducción a la robótica médica

La robótica es una de las mayores aliadas en todos los sectores laborales hoy en día. Con el avance de la tecnología y la automatización, la robótica se está viendo más y más incluida en cualquier proceso que necesite una eficacia y seguridad óptimas. Dentro del campo de la medicina, se ha logrado implementar máquinas automatizadas que han ayudado al desarrollo de funciones como la cirugía, la asistencia sanitaria o terapia de rehabilitación [1].

El objetivo que persigue la robótica dentro de la medicina es el de facilitar y asistir en el trabajo de los profesionales en este campo, de forma que los beneficiados consigan una mayor precisión en procesos como las operaciones o minimizaciones en las limitaciones del ser humano.

La robótica médica tiene sus orígenes en los años 80, cuando se desarrolló el sistema PUMA 560, el primer brazo robótico asistente en una neurocirugía, el cuál fue desarrollado principalmente para realizar operaciones telemáticamente para la NASA. Más tarde apareció el sistema AESOP, fabricado por Computer Motion Incorporated, que fue el primer robot aprobado por la FDA (Food and Drug Administration). El cirujano controla el manipulador a través de comandos de voz. Los comandos se graban previamente, de forma que cada cirujano tiene una tarjeta de sonido personalizada. Esta misma empresa desarrolló otro robot, el sistema Zeus, con el cuál se inició la telerobótica en la cirugía robótica. Por último, uno de los avances más importantes en la robótica médica es la aparición del robot Da Vinci en el año 2000, que fue aprobado por la FDA como sistema apto para realizar cirugías de complejidad media [2]. La evolución de los sistemas robóticos quirúrgicos puede verse en la Figura 1.

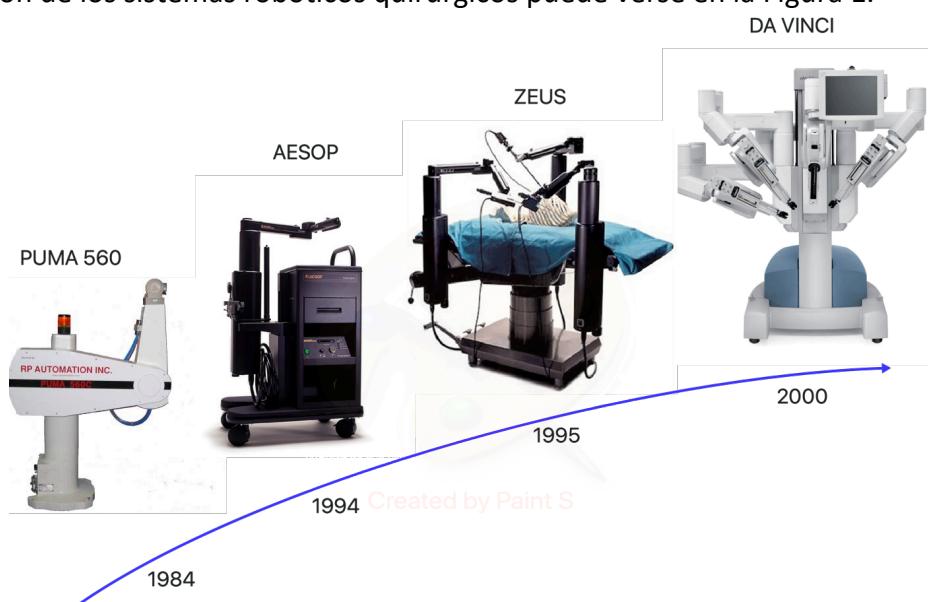


Figura 1. Evolución de la robótica quirúrgica.

El sistema Da Vinci (Figura 2) está formado por diferentes elementos, como son la consola maestra, el robot esclavo, su instrumental y el sistema de obtención de imagen. Se trata de un robot Maestro-Esclavo, de forma que el cirujano indica todo lo que debe hacer el robot para que este lo repita. La consola maestra es el sistema de control, donde el cirujano realiza los movimientos que se mandan al esclavo. En esta consola, por un lado el cirujano recibe las imágenes en tres dimensiones que le proporcionan el sistema de obtención de imagen, y por otro lado tiene unos dedos manipuladores que le permiten describir el movimiento a realizar. El robot esclavo está compuesto de 3 o 4 brazos, de los cuales uno contiene la cámara por la cuál el cirujano recibe las imágenes. Los otros brazos se encargan de manejar los instrumentales quirúrgicos y de realizar los movimientos que reciben de la consola maestra. Los instrumentales quirúrgicos cuentan con un sistema llamado EndoWrist, cuya punta reproduce todos los movimientos recibidos por el ordenador. Esta herramienta cuenta con 7 grados de libertad, lo que proporciona una gran destreza a la hora de realizar cualquier cirugía. Por último, el sistema de obtención de imagen se trata de una cámara doble, que recoge información por dos canales que permiten formar una imagen estereoscópica, proyectada en dos monitores, uno para cada ojo. Para darle al cirujano una sensación de estar inmerso, se utiliza el sistema de “caja de espejos” [3].



Figura 2. Sistema Da Vinci.

Dentro de la robótica médica, el campo en el que se desenvuelve este trabajo de fin de grado es la robótica quirúrgica aplicada a la Cirugía mínimamente Invasiva (CMI). La CMI tiene como principal tarea realizar cirugías minimizando al máximo el daño, realizando pequeñas incisiones que tienen numerosas ventajas frente a una cirugía convencional. Estas incisiones son principalmente para la cámara, la cuál permite que el

cirujano reciba realimentación visual del interior del paciente, y para el instrumental médico [4].

Todos los procedimientos de CMI implican una reducción de la agresión quirúrgica. Las principales ventajas que presenta son:

- Reducción de la inflamación asociada con la cirugía y la mejoría en la respuesta inmunológica.
- Disminución del dolor postoperatorio principalmente por la falta de incisiones quirúrgicas grandes.
- Menos complicaciones en la herida quirúrgica. Las heridas son más pequeñas, por lo que la cicatrización se convierte en un proceso más rápido.
- Disminución de la estancia en el hospital debido a una recuperación más sencilla.

Pero a pesar de las numerosas ventajas que presenta este procedimiento, la efectividad de la laparoscopia en cirugías complicadas está por probar. En ciertas situaciones de mucha complejidad, el cirujano puede llegar a verse limitado (Figura 3), debido a que los instrumentos laparoscópicos no tienen los suficientes grados de libertad, ni sistemas de retroalimentación de fuerza, que pueden complicar la cirugía hasta el punto de que el cirujano prefiera realizar una operación abierta.

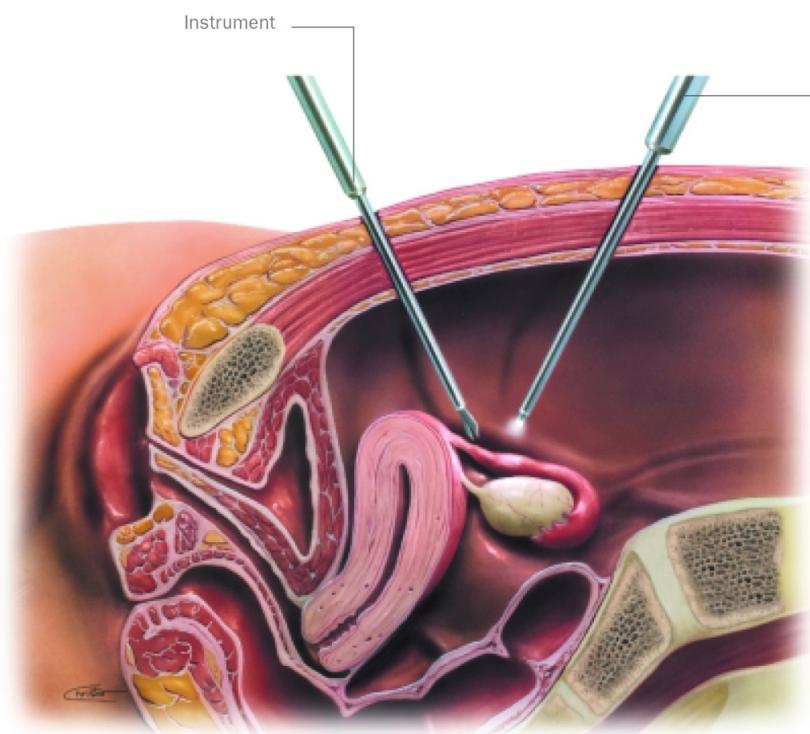


Figura 3. Las limitaciones del punto de fulcro exigen una posición y orientación específicas

La aparición de los robots quirúrgicos es la principal solución a los problemas de la CMI, ya que han conseguido reducir las principales limitaciones, así como expandir las capacidades del cirujano. Con la inclusión de la robótica en este tipo de cirugía, se ha

hecho posible la incorporación de ciertas tecnologías como la visión tridimensional y la retroalimentación de fuerza para crear una sensación de tacto, entre otras.

El principal problema que va a abarcar este trabajo de fin de grado es el de la libertad de movimiento de las herramientas dentro del abdomen del paciente en una CMI. El puerto de entrada del instrumental quirúrgico se llama punto de fulcro, y al tener que pasar por ahí, las herramientas experimentan una limitación a la hora de realizar cualquier movimiento. Durante la intervención, la herramienta va a tener tan solo 4 grados de libertad (Figura 4), que son dos rotaciones entorno al punto de fulcro, una rotación entorno al eje principal de la herramienta, y el desplazamiento de la herramienta a lo largo de dicho eje. Por tanto, cuando se quiera dirigir la punta de esta a una posición, no se puede hacer libremente, sino que hay que jugar con los grados de libertad disponibles para realizar el movimiento necesario para alcanzar dicha posición. En la Figura 4 se muestra un esquema de cuáles son dichos grados de libertad.

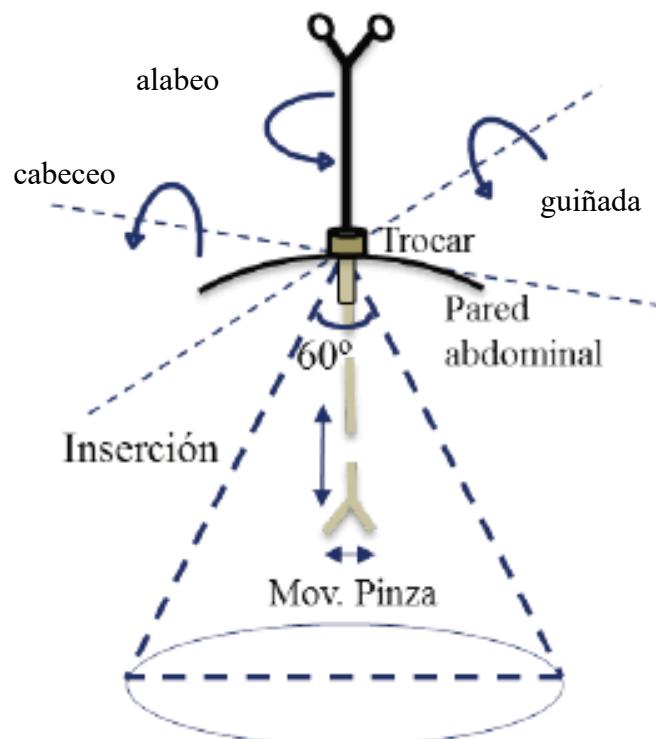


Figura 4. Grados de libertad de la herramienta.

1.2 Objetivos del TFG

El objetivo principal de este Trabajo de Fin de Grado es implementar un esquema de control de movimiento de un brazo robótico, en concreto el UR3 de Universal Robots, para el manejo de una herramienta laparoscópica, teniendo en cuenta las restricciones de movimiento impuestas por el punto de fulcro. Este esquema de control se implementará en el lenguaje de programación Python, y se integrará en el sistema operativo ROS.

Para poder alcanzar este objetivo, se han planteado unos objetivos específicos que dividen el trabajo de la siguiente forma:

- 1. Estudio teórico del esquema de control de movimiento de una herramienta laparoscópica alrededor de un punto de fulcro:** Como se menciona anteriormente, para realizar un determinado movimiento de la herramienta laparoscópica hay que considerar unas limitaciones impuestas por el punto de fulcro. Por tanto, se ha tenido que hacer un estudio para hallar las ecuaciones necesarias a la hora de calcular la posición y orientación del efecto final del robot para que la punta de la herramienta alcance su posición objetivo.
- 2. Implementación del esquema de control desarrollado en Python e integración en ROS:** Una vez realizado el estudio anterior y teniendo las ecuaciones necesarias desarrolladas, se van a implementar en ROS. Este sistema operativo permite programar tanto en lenguaje C++ como en Python, para este trabajo se va a realizar la programación con Python. La implementación del código en ROS nos va a permitir que sea un código genérico, en otras palabras, cualquier robot capaz de utilizar ROS va a poder ejecutar este programa y realizar la tarea para la cual se ha realizado el código. No obstante, este TFG se ha realizado con el objetivo de implementarlo en el robot UR3 de Universal Robots, de forma que las pruebas finales y simulaciones se realizarán sobre este.
- 3. Implementación de un interfaz de usuario que permita tanto definir ciertos parámetros del sistema, como comandar los movimientos del robot:** Se va a diseñar un interfaz que nos facilitará algunas tareas como la de definir unos parámetros necesarios para el comienzo de la tarea. Este interfaz de usuario va a dividirse en dos partes, una configuración en la que vamos a poder declarar dichos parámetros, y una parte de movimiento, que nos va a permitir indicar el movimiento que queremos que realice nuestro robot, de forma que le mandará las indicaciones una vez introduzcamos estos valores.
- 4. Desarrollo de una serie de experimentos para validar el trabajo desarrollado:** Para verificar el correcto funcionamiento del código, vamos a realizar una serie de experimentos a través de un simulador, que nos permitirá saber cuáles son los movimientos que realizará nuestro robot al indicarle la posición a la que queremos que llegue la punta de la herramienta laparoscópica.

Para llevar a cabo estos objetivos, se han realizado algunas tareas adicionales que nos pueden facilitar parte del trabajo, que son:

- **Estudio de librerías de Python y ROS:** Para realizar las operaciones requeridas se va a hacer un estudio sobre qué librerías existen en Python que pueden facilitarnos algunos cálculos, como por ejemplo la conversión de la forma de orientación que recibimos de nuestro robot en otra forma necesaria para realizar los cálculos, o la librería que nos permite crear una interfaz de usuario.
- **Uso de Simulador y URXdriver para comprobar los resultados:** Por un lado tenemos la herramienta URXdriver facilitada por la UMA, que va a permitir visualizar cada movimiento realizado por el robot. Esta herramienta se va a conectar al simulador a través de su dirección IP, de forma que toda la información tendrá que ser enviada o recibida por los topics de la herramienta, que tendrá el mismo aspecto que el brazo robótico que usa el simulador. Por otro lado tenemos el simulador de Universal Robots, que se instalará en una máquina virtual y nos va a servir para comprobar los resultados.

1.3 Estructura del TFG

La organización que se ha utilizado para desarrollar el proyecto es la siguiente:

- **Capítulo 2, Control de movimiento alrededor de un punto de fulcro:** En este capítulo se realiza un estudio teórico de las ideas utilizadas para desarrollar el proyecto. Se exponen todas las bases teóricas que han servido para realizar cualquier cálculo necesario.
- **Capítulo 3, Implementación:** En este capítulo se va a realizar una explicación de la forma de implementar el código en ROS, usando el lenguaje de Python. Se muestra la distribución de cada nodo, así como la explicación del objetivo que se consigue con cada una de las partes.
- **Capítulo 4, Experimentación y resultados:** En este capítulo se exponen todas las pruebas realizadas para comprobar el correcto funcionamiento. Con todas las demostraciones realizadas con los simuladores que permiten conocer si los resultados son los esperados.
- **Capítulo 5, Conclusiones y futuros trabajos:** En este capítulo se hace un pequeño resumen de lo conseguido mediante el trabajo, recapitulando sobre la manera de conseguir los objetivos, y proponiendo algunas mejoras que se podrían implementar.

2 CONTROL DE MOVIMIENTO ALREDEDOR DE UN PUNTO DE FULCRO

En este capítulo se va a desarrollar el planteamiento teórico realizado para conseguir el objetivo del trabajo.

2.1 Esquema de control

Este apartado presenta la idea principal del trabajo, la base sobre la cuál se ha ido realizando cada avance. Lo que representa este esquema es la forma en la que se genera la posición y la orientación deseadas para nuestro manipulador, P_d y V_d respectivamente, para que la punta de la herramienta se mueva un determinado valor ΔP . El control de movimiento, encargado de calcular las variables requeridas por el robot, recibe los siguientes parámetros:

- **ΔP :** Incremento de posición deseado de la herramienta. Es un vector de 3 componentes, referentes al incremento que se desea realizar en los ejes X, Y, Z.
- **D:** Longitud de la herramienta. Como este trabajo se realiza de forma que se pueda implementar en diferentes manipuladores, esta es una variable que puede variar en función del robot.
- **P_f :** Distancia del efecto final al punto de fulcro. Este parámetro es muy importante, ya que sirve para calcular la posición del punto de fulcro, que será el orificio por donde se introduce la herramienta. Es fundamental que se introduzca el valor correcto ya que el resto de los cálculos dependerán de dicha posición. Se calcula al inicio y permanece constante durante todo el proceso.
- **P:** Posición actual del robot. Es proporcionada con respecto al sistema de referencia global, que es el de la base del robot.
- **V:** Orientación actual del robot. Recibida en forma de vector de rotación.

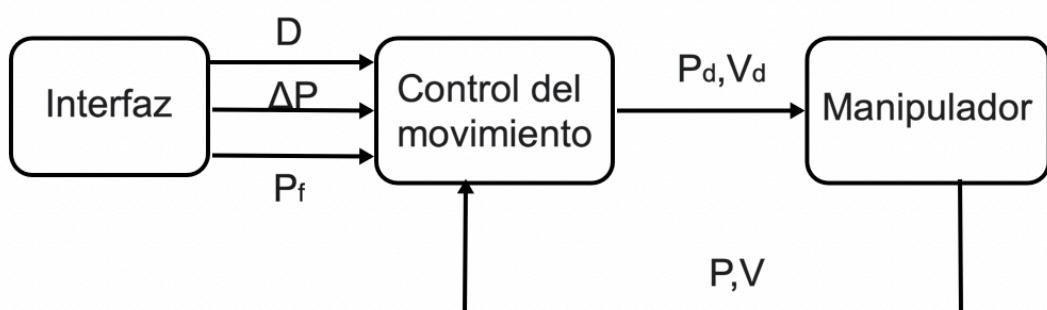


Figura 5. Esquema de control del proyecto.

Los parámetros ΔP , D , P_f se reciben desde la interfaz de usuario, mientras que P y V son valores del manipulador. Este esquema de control puede observarse en la Figura 5.

2.1 Representación de la posición

Para la representación de la posición en robótica se suelen usar diferentes coordenadas, que son:

- **Coordenadas cartesianas (x, y, z):** Los puntos se sitúan según la distancia de cada coordenada al origen de los ejes.
- **Coordenadas polares:** Las polares son un sistema que representa la posición mediante dos coordenadas, la coordenada radial (ρ) que expresa la distancia desde la posición hasta el origen, y la coordenada angular (θ) que expresa el ángulo desde el eje hasta la posición.
- **Coordenadas cilíndricas:** Las coordenadas cilíndricas representan la posición mediante tres parámetros, ρ expresa la distancia entre la posición y el eje Z, θ expresa el ángulo formado con el eje X, y Z expresa la altura.

2.2 Representación de la orientación

En este apartado se van a describir los tipos de representación de la orientación que se ha usado para nuestro manipulador, así como las ecuaciones necesarias para poder transformar la orientación de una representación a otra.

2.2.1 Matriz de rotación

Para representar la orientación de un cuerpo en el espacio en forma de matriz de rotación, se debe asignar un sistema de coordenadas referente a dicho cuerpo y describir la relación espacial entre este sistema de coordenadas y un sistema de coordenadas de referencia. Para describir el sistema de coordenadas del cuerpo, se escriben los vectores unitarios referentes a sus ejes principales en función del sistema de coordenadas de referencia. Como resultado obtenemos tres vectores unitarios, uno para cada eje del sistema de coordenadas del cuerpo. Si colocamos estos tres vectores como columnas de una matriz obtendremos la matriz de rotación, de dimensión 3x3 [5].

Como ejemplo, consideramos un sistema de referencia en el origen de los ejes X, Y, Z, (O_{xyz}) y como sistema de coordenadas del cuerpo el sistema O_{abc} , y unos vectores unitarios x' , y' y z' en dirección de los ejes a , b , c respectivamente. La matriz de rotación que expresa la orientación del sistema O_{abc} respecto del sistema de referencia sería la siguiente:

$${}_{OXYZ}R_{Oabc} = \begin{bmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{bmatrix} \quad [1]$$

Las matrices de rotación básicas permiten realizar rotaciones de vectores alrededor de los ejes X, Y, Z. Las siguientes matrices expresan la rotación de un ángulo θ de un vector sobre los ejes:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \quad [2]$$

$$R_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad [3]$$

$$R_z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad [4]$$

Las matrices de rotación pueden componerse para representar la orientación de un cuerpo que realiza varias rotaciones. Al no ser conmutativo el producto de las matrices, debe tenerse en cuenta el orden de las rotaciones.

2.2.2 Ángulo-eje y Vector de rotación

Como se ha descrito anteriormente, una matriz de rotación es un vector que contiene nueve elementos. Estos elementos están sujetos a las seis restricciones de la ortogonalidad y la norma, de forma que solo quedan tres grados de libertad, lo que quiere decir que si tenemos tres de los elementos de la matriz de rotación, los otros seis pueden calcularse a través de esas restricciones. En ciertos casos, las matrices de rotación pueden ser inconvenientes, siendo preferible una representación de la orientación más simple.

La forma más sencilla de representarla es el ángulo-eje, mostrado gráficamente en la Figura 6, que está basada en el teorema de Euler, y que declara que cada rotación

se puede describir en un solo eje de rotación (e) que indica la dirección, y ángulo (θ) que describe la magnitud de la rotación alrededor de este. Debido a que el eje debe ser unitario, solo tiene dos grados de libertad, mientras que añadiendo el ángulo de giro se consiguen los tres grados de libertad de esta representación de la orientación.

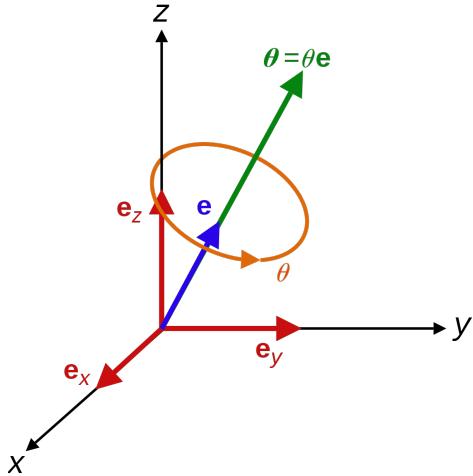


Figura 6. Representación del Ángulo-Eje y Vector de rotación.

Una forma más compacta de expresar el ángulo-eje es el vector de rotación, un vector de tres dimensiones no normalizado, cuya dirección indica el eje, y cuya longitud es θ . Esta representación contiene tres valores escalares, los cuales son los tres grados de libertad de esta forma de expresar la orientación, obtenidos de multiplicar el ángulo y el eje.

$$r = \theta \hat{e} \quad [5]$$

El problema que tiene esta representación es que el conjunto de todos los vectores de rotación forma una bola en tres dimensiones de radio π , de forma que dos puntos distintos de la esfera r y $-r$, con norma π representan la misma rotación de 180 grados (Figura 7) [6].

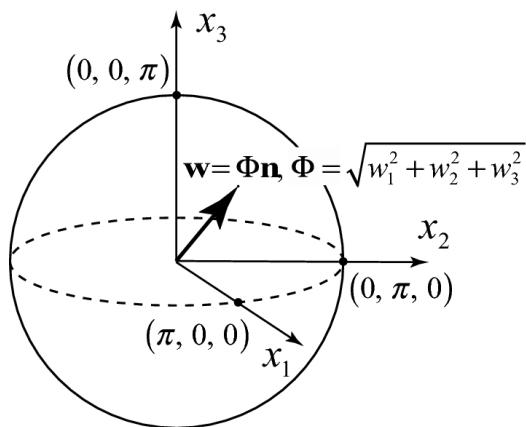


Figura 7. Problema de la representación del Vector de rotación.

2.2.3 Ángulos de Euler

Es un conjunto de tres coordenadas angulares que permiten describir la orientación de un sistema de referencia de ejes móviles respecto de otro sistema de ejes fijos.

Considerando los sistemas de coordenadas ABC y XYZ, de los cuáles podemos obtener la línea de nodos, que aparece de la intersección de los planos AB y XY, y siendo su origen común, se puede representar la posición de un sistema de coordenadas en relación con el otro con los ángulos (α , β y γ), mostrada en la Figura 8, definiéndolos como:

- α es el ángulo formado entre el eje A y la línea de nodos.
- β es el ángulo formado entre el eje C y el eje Z.
- γ es el ángulo formado entre la línea de nodos y el eje X

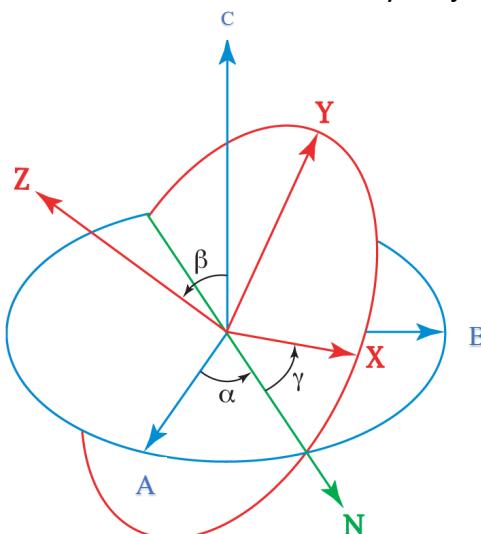


Figura 8. Representación de la forma de definir los ángulos de Euler.

2.3 Conversión entre las distintas representaciones de la orientación

En el apartado 2.2.2 se explica la forma de pasar de Ángulo-Eje a Vector de rotación. A pesar de la sencillez de dicha conversión, para realizar la conversión inversa se deben tener en cuenta ciertas cosas. Se dice que el eje debe ser un vector unitario, por lo que la norma debe valer 1. Al estar el ángulo multiplicado en cada uno de los componentes del vector, al calcular la norma debemos obtener como resultado el ángulo [6]. Sea un vector de rotación r:

$$r = (\theta v_x, \theta v_y, \theta v_z) \quad [6]$$

Si calculamos su norma tenemos que:

$$\sqrt{(\theta v_x)^2 + (\theta v_y)^2 + (\theta v_z)^2} =$$

$$\sqrt{\theta^2(v_x^2 + v_y^2 + v_z^2)} =$$

$$\sqrt{\theta^2} = \theta$$

Por tanto;

$$\theta = \sqrt{(r_x)^2 + (r_y)^2 + (r_z)^2} \quad [7]$$

Para calcular el eje teniendo el ángulo, no hay más que dividir cada componente del vector de rotación entre el ángulo.

$$v = \left(\frac{r_x}{\theta}, \frac{r_y}{\theta}, \frac{r_z}{\theta} \right) \quad [8]$$

Conversión Matriz de rotación a Eje-Ángulo

Para transformar la matriz de rotación en una representación de eje-ángulo, se determinará primero el eje, y posteriormente se calculará el ángulo [7].

Cálculo del eje

Considerando una matriz de rotación R de tres dimensiones y un vector v paralelo al eje de rotación, se cumple que $Rv = v$ debido a que una rotación del vector sobre el eje de rotación debe resultar en el mismo vector.

La anterior ecuación puede ser reformulada como:

$$Rv = Iv \rightarrow (R - I)v = 0 \quad [9]$$

Una vez considerada la ecuación, se puede determinar el eje de rotación como:

$$\begin{aligned} 0 &= R^T 0 + 0; \\ 0 &= R^T(R - I)v + (R - I)v; \\ 0 &= (R^T R - R^T + R - I)v; \\ 0 &= (I - R^T + R - I)v; \\ 0 &= (R - R^T)v; \end{aligned}$$

De esta última ecuación, podemos concluir en que si el producto vectorial de la matriz con el vector v debe dar 0;

$$(R - R^T) = [v], \quad [10]$$

Ya que para que el producto vectorial sea nulo, debemos conseguir un vector v que al multiplicarlo por dicha matriz resulte 0. De este modo ya podríamos calcular el eje de rotación.

Considerando:

$$R = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}, \quad [11]$$

Tenemos entonces que:

$$v = \begin{bmatrix} h - f \\ c - g \\ d - b \end{bmatrix} \quad [12]$$

Cálculo del ángulo

Para calcular el ángulo, tan solo hay que hallar la traza de la matriz de rotación, y teniendo esta seguir la ecuación:

$$Tr(R) = 1 + 2\cos(\alpha) \quad [13]$$

De esta forma tendríamos todos los componentes de la representación eje-ángulo, calculados a partir de la matriz de rotación.

Conversión Eje-Ángulo a Matriz de rotación

Para conseguir la matriz de rotación a partir de la representación de eje-ángulo, consideramos un vector $v = (v_x, v_y, v_z)$ unitario y un ángulo α que genera la rotación alrededor del vector v . A partir de estos parámetros, la matriz se calcula como:

$$R = \cos(\alpha) I + \sin(\alpha) [v] + (1 - \cos(\alpha))(vv^T) \quad [14]$$

Donde I es la matriz identidad, vv^T tiene como resultado una matriz de dimensión 3x3, y $[v]$ es la matriz producto vectorial:

$$[v] = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix} \quad [15]$$

Conversión Ángulos de Euler a Matriz de rotación

Para este trabajo se utiliza la conversión de ángulos de Euler ZYZ a matriz de rotación, que se consigue mediante la composición de tres matrices básicas conociendo los ángulos de rotación α , β y γ . La matriz de rotación se calcula como:

$$R_{ZYZ}(\alpha, \beta, \gamma) = R_Z(\alpha) \cdot R_Y(\beta) \cdot R_Z(\gamma) \quad [16]$$

Obteniendo:

$$R_{ZYZ} = \begin{bmatrix} c_\alpha c_\beta c_\gamma - s_\alpha s_\gamma & -c_\alpha c_\beta s_\gamma - s_\alpha c_\gamma & c_\alpha s_\beta \\ s_\alpha c_\beta c_\gamma + c_\alpha s_\gamma & -s_\alpha c_\beta s_\gamma + c_\alpha c_\gamma & s_\alpha s_\beta \\ -s_\beta c_\gamma & s_\beta s_\gamma & c_\beta \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \quad [17]$$

Conversión Matriz de rotación a Ángulos de Euler

Para obtener los ángulos a partir de la matriz de rotación, utilizamos las ecuaciones:

$$\alpha = \arctan\left(\frac{R_{23}}{\sin\beta} / \frac{R_{13}}{\sin\beta}\right) \quad [18]$$

$$\beta = \arctan(\pm \sqrt{(R_{31}^2 + R_{32}^2) / R_{33}}) \quad [19]$$

$$\gamma = \arctan\left(\frac{R_{32}}{\sin\beta} / \frac{-R_{31}}{\sin\beta}\right) \quad [20]$$

2.4 Estimación del punto de fulcro

Como primer paso para calcular la posición que debe alcanzar nuestro manipulador, tenemos que conocer la posición del punto de fulcro. La correcta estimación de dicho punto será fundamental para el resto del trabajo, ya que sin él, no podremos mover el robot a ninguna pose. De forma genérica, la ecuación que se va a seguir para su cálculo es la siguiente:

$$P_f = P + \rho_0 D_T \bar{z} \quad [21]$$

Donde:

- P_f será la posición del punto de fulcro.
- P es la posición actual del efecto final del robot, que nos la proporcionará el propio manipulador en coordenadas cartesianas, respecto al sistema de referencia de la base.
- D_t es la longitud de la herramienta, la cuál puede variar en función de la herramienta.
- \bar{z} es el eje Z del efecto final. La orientación del manipulador es un valor que se recibirá de él, pero será recibido en representación de Vector de rotación, y por tanto tendremos que realizar la conversión apropiada.
- ρ_0 es un parámetro que indica la posición del punto de fulcro con relación a la longitud de la herramienta y su valor estará entre 0 y 1. Por ejemplo, un valor de 0.5 indica que el punto de fulcro se encuentra en la mitad de la herramienta. Este parámetro se calcula de forma manual midiendo, en la posición inicial del robot, la distancia del efecto final al punto de fulcro.

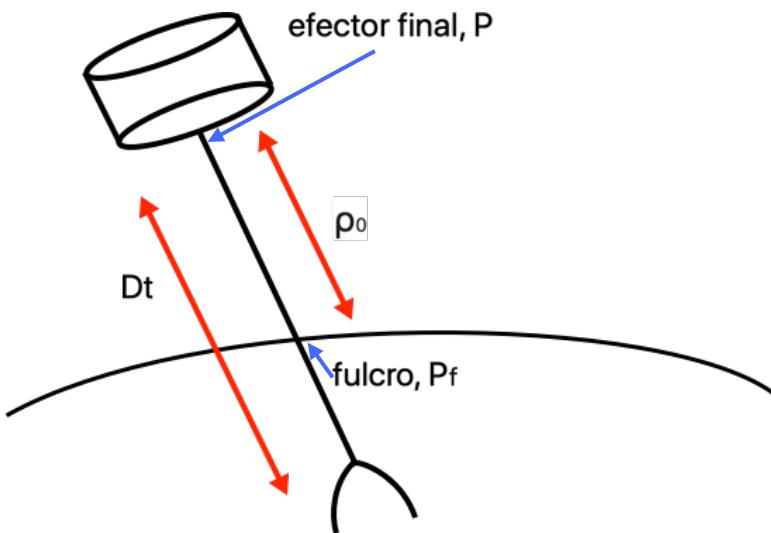


Figura 9. Demostración de la forma de calcular el punto de fulcro.

En la Figura 9 se muestra un ejemplo del manipulador, con todas las variables que intervienen en la ecuación. Por tanto, teniendo la posición del efecto final, sabiendo la orientación, y la distancia relativa con respecto a la longitud de la herramienta, podemos hacer la estimación que nos dará la posición del punto de fulcro, el cuál almacenaremos en una variable en coordenadas cartesianas.

El procedimiento seguido para conseguir todos los parámetros de la ecuación es el siguiente:

1. Se coloca el robot en una posición concreta, va a ser la posición inicial desde la cual se va a realizar toda la operación. Tras ello, se reciben los parámetros D_T y la distancia que hay desde el efecto final hasta el punto de fulcro. Este último parámetro nos servirá para calcular ρ_0 .
2. Se recibe la pose del robot, tanto posición como orientación, y se separa en dos variables distintas P y O, respectivamente.
3. La orientación se recibe en forma de vector de rotación, pero para conseguir el eje Z del manipulador se necesita la orientación en forma de matriz de rotación, por tanto, se convierte el vector de rotación en eje-ángulo, como se describe en el apartado anterior, y de esta representación se convierte a matriz de rotación. De la matriz extraemos la última fila, que hace referencia a la dirección del eje Z del manipulador.
4. Se calcula el valor de ρ_0 con los dos valores recibidos en el primer paso.
5. Se realiza el cálculo de la posición del punto de fulcro teniendo el valor de todos los parámetros necesarios.

2.5 Movimiento alrededor del punto de fulcro

Una vez conocemos el punto de fulcro, se realizará el cálculo de la posición y orientación que debe adoptar nuestro manipulador con el objetivo de que la punta de la herramienta alcance una posición determinada. Este paso se repetirá cada vez que se quiera realizar un nuevo movimiento de la punta, sin embargo, el cálculo de la posición del punto de fulcro se realizará solo la primera vez, ya que será constante durante el proceso.

El procedimiento seguido para alcanzar el objetivo mostrado en la Figura 10 es el siguiente:

1. Se calcula la posición de la punta de la herramienta actual, que es la posición a la que le sumaremos los incrementos deseados. Teniendo la posición actual del manipulador, la longitud de la herramienta y la dirección del eje Z del efecto final, la posición de la punta de la herramienta se calcula como:

$$P_t = P + D_T \bar{z} \quad [22]$$

Donde:

- P_t será la posición de la punta de la herramienta, en coordenadas cartesianas.
 - P es la posición actual del robot, también en coordenadas cartesianas.
 - D_T es la longitud de la herramienta.
 - \bar{z} es el eje Z de la orientación del efecto final.
2. Recibimos el incremento de posición que queremos realizar en cada una de las direcciones para añadírselo a la posición de la herramienta ya calculada.

$$P_{tn} = P_t + \Delta P \quad [23]$$

Donde:

- P_{tn} será la nueva posición de la herramienta.
 - ΔP es el incremento de posición recibido..
3. Calculamos el vector que une la nueva posición de la punta con el punto de fulcro. Este será el nuevo vector de dirección de eje Z que tendrá nuestra herramienta.

$$\bar{z}_t = P_f - P_{tn} \quad [24]$$

Donde:

- \bar{z}_t es el vector de dirección.
 - P_f es el punto de fulcro calculado anteriormente.
 - P_{tn} es la posición a la que irá la herramienta.
4. Se calcula la distancia desde el punto de fulcro hasta la nueva posición del efecto final. Sabiendo la longitud total de la herramienta, y la distancia que hay desde el punto de fulcro a la punta de esta, tenemos:

$$\rho_n = D_T - \|\bar{z}_t\| \quad [25]$$

Donde:

- ρ_n será la distancia deseada.

- D_T es la longitud de la herramienta, obtenida anteriormente.
 - $\|\bar{z}_t\|$ es el módulo del vector dirección.
5. Una vez tenemos esta distancia, podemos calcular la nueva posición del efecto final. Si desde el punto de fulcro sumamos la distancia hasta el efecto final en la correcta dirección tenemos:

$$P_n = P_f + \rho_n \bar{z}_t \quad [26]$$

Donde:

- P_n será la nueva posición del efecto final, en coordenadas cartesianas.
- P_f posición del punto de fulcro, ya calculada.
- ρ_n distancia del fulcro al efecto final.
- \bar{z}_t nueva dirección del manipulador.

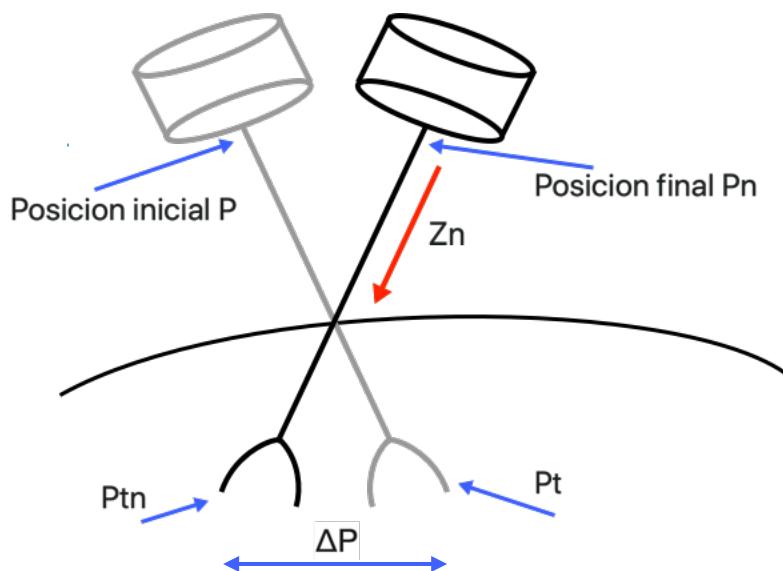


Figura 10. Posición final del efecto final tras realizar el movimiento desde la posición inicial

Para realizar el cálculo de la orientación de la herramienta se va a crear primero una matriz de rotación, que se transformará en representación Eje-ángulo, y finalmente se convertirá en vector de rotación para poder mandarlo al manipulador. Para crear la

matriz se usarán los ángulos de Euler ZYZ, los cuáles calcularemos a través de nuestra nueva dirección del eje Z del manipulador.

Al igual que para obtener la orientación inicial del robot se coge solo la última fila de la matriz, la cuál hace referencia a la dirección del eje Z, el cálculo de la matriz de rotación actual se va a realizar a partir de nuestro nuevo eje, siendo este la última fila de nuestra nueva matriz. Los pasos para conseguir la orientación son los siguientes:

1. Convertimos nuestro vector de dirección \bar{z}_t en unitario, ya que para que forme parte de nuestra matriz de rotación, cada uno de los vectores deben ser unitarios:

$$M_{zt} = \sqrt{z_{tx}^2 + z_{ty}^2 + z_{tz}^2} \quad [27]$$

$$z_{nn} = \left(\frac{z_{tx}}{M_{zt}}, \frac{z_{ty}}{M_{zt}}, \frac{z_{tz}}{M_{zt}} \right) \quad [28]$$

2. Se calculan los ángulos de Euler. Para este apartado hemos elegido los ángulos de Euler ZYZ, ya que con esta precisa representación es posible conseguir dos de los tres ángulos (β y γ) a partir de la última fila de la matriz de rotación, que equivale a nuestro nuevo eje Z (z_{nn}). El último ángulo, α no se puede calcular a partir del nuevo eje Z, sin embargo, al ser un ángulo que representa la rotación con respecto al eje X, no importa el valor que tenga. Por tanto:

$$\beta = \arctan (\pm \sqrt{(z_{nn_x}^2 + z_{nn_y}^2)} / z_{nn_z}) \quad [29]$$

$$\gamma = \arctan \left(\frac{z_{nn_y}}{\sin \beta} / \frac{-z_{nn_x}}{\sin \beta} \right) \quad [30]$$

$$\alpha = 0 \quad [31]$$

3. Se crea la matriz de rotación a partir de los tres ángulos obtenidos, tal y como se describe en la ecuación 17.
4. Se convierte la representación de matriz de rotación en Eje-ángulo, y de esta a vector de rotación para mandarla (ecuación 4).

3 IMPLEMENTACIÓN

3.1 Introducción a ROS

Para la realización de este trabajo se ha utilizado ROS, un middleware robótico que sirve para el desarrollo de software de robots.

ROS (Robotic Operative System) se desarrolló en 2007 por el Laboratorio de Inteligencia Artificial de Standford. Este software está basado en una colección de herramientas y librerías necesarias para el desarrollo de software para la robótica. Aunque no es un sistema operativo como tal, es capaz de controlar dispositivos de bajo nivel, proporcionar servicios de abstracción del hardware y comunicar procesos entre sí entre otras muchas [8].

ROS se estructura de forma que el procesamiento se realice a través de unos nodos, que se encargan de recibir, mandar y multiplexar datos de sensores, estados y actuadores entre ellos. Se necesita una máquina tipo Unix como Linux o Mac Os X para poder ejecutar este software

La filosofía de ROS está distribuida en cuatro principios:

1. **Punto-a-punto:** Esta arquitectura permite que los nodos que componen el robot manden y reciban información directamente con otro nodo.
2. **Multilenguaje:** El marco ROS puede ser implementado en varios lenguajes de programación moderno, como son Python y C++, aunque actualmente se están desarrollando bibliotecas en Java.
3. **Basado en herramientas:** ROS tiene un diseño de microkernel, que usa una gran cantidad de herramientas para construir y ejecutar todos sus nodos.
4. **Ligero:** Los algoritmos se empaquetan en ejecutables independientes reutilizables.

3.2 Partes de ROS

Para entender cómo funciona ROS es necesario explicar de qué se compone su jerarquía.

3.2.1 Master

Se encarga tanto de registrar los nodos como de crear las conexiones entre ellos, una vez los une, los nodos se comunican entre ellos sin necesidad de pasar por el master. Busca nodos publicadores y suscriptores de topics y realiza la conexión necesaria para que se comuniquen entre ellos.

Un nodo quiere publicar un dato por un topic, por tanto avisa al master de que va a publicar, y aunque no haya ningún nodo suscrito a ese topic, va a seguir publicando.

Otro nodo indica al master de que quiere suscribirse al topic, por tanto, el master informa a ambos nodos de la existencia del otro para que puedan mandar y recibir información entre ellos a través de dicho topic.

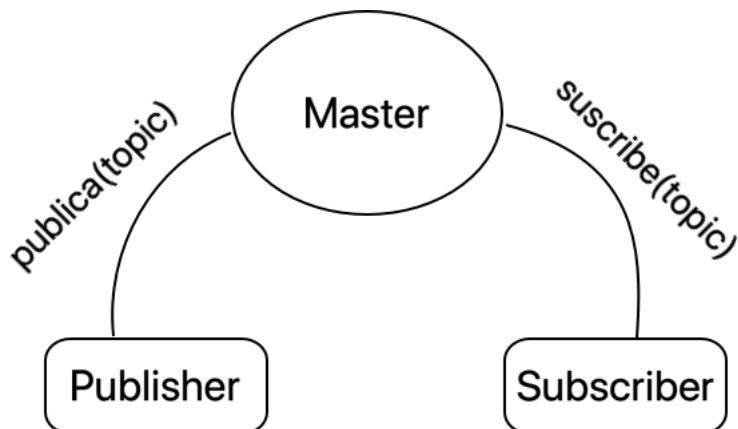


Figura 11. Registro de nodos.

En la Figura 11 se muestra un ejemplo de dos nodos que se registran e informan al master de que quieren publicar y suscribirse a un topic determinado. En la Figura 12, el master crea la conexión entre los dos nodos para que puedan comunicarse entre ellos.

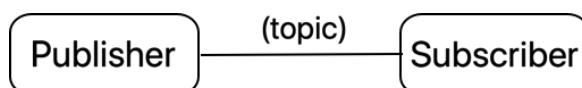


Figura 12. Comunicación entre nodos.

3.2.2 Topics

Es el método más común para intercambiar mensajes entre los nodos. Se puede entender como un bus por el que se intercambian mensajes, cualquier nodo puede

publicar mensajes en dicho bus siempre que sea del tipo correcto. Un nodo quiere publicar un tipo de mensaje, y si el topic por el que va a publicar no ha sido creado anteriormente, se crea al publicar por él. En este proyecto, los topics que van a ser utilizados son:

- **UR3_1/outputs/pose:** Este topic al que nos vamos a suscribir contiene la información sobre la posición y orientación del robot en cada momento. Cada vez que nuestro manipulador se mueve, devuelve un array de 6 elementos por este topic, cuyos tres primeros valores serán la posición del efecto final del eje X, Y, Z respectivamente y con respecto al sistema de referencia global. Los últimos tres valores hacen referencia a la orientación del efecto final, representado en vector de rotación.
- **UR3_1/inputs/move_pose:** Este topic va a ser el canal por el que vamos a publicar la información sobre la pose que queremos que adopte nuestro robot. Su estructura es igual a la del topic anterior, reservando los tres primeros valores del array a la posición, y los tres últimos para la orientación.
- **configuración:** Este topic establece una conexión entre nuestros dos nodos. El nodo interfaz publica por este canal los datos de configuración que necesita nuestro nodo movimiento para realizar los cálculos.
- **movimiento:** Con la misma idea que el topic anterior, nuestro nodo de movimiento va a suscribirse esperando unos comandos de movimiento que van a ser publicados desde el nodo interfaz, y que van a ser los incrementos de posición que se desea que realice la punta de la herramienta.

En la Figura 13 se muestra un esquema de la conexión entre los nodos, mediante los topics utilizados.

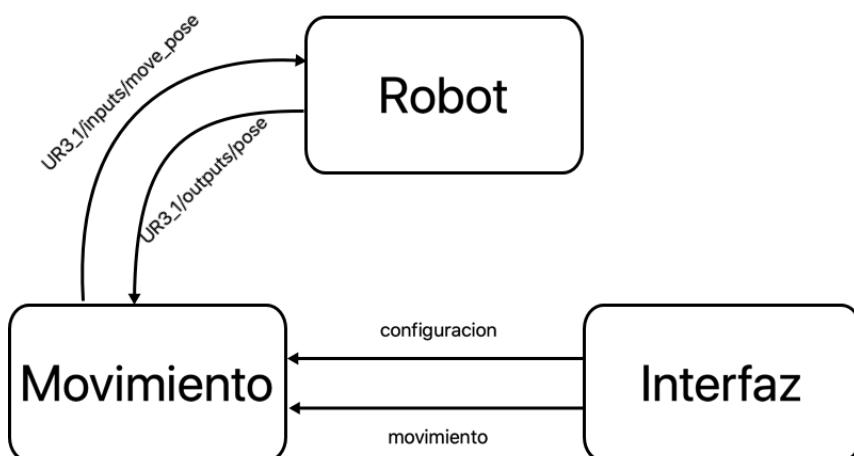


Figura 13. Comunicación entre los nodos activos a través de los topics.

3.2.3 Nodos

Los nodos son procesos con una función específica. Como se menciona en el apartado 3.2.1, se registran y tras ello se comunican con otros nodos a través de topics o servicios, formando módulos independientes. En nuestro caso, vamos a tener dos nodos, nodo interfaz y nodo movimiento, los cuales van a realizar una función específica para el desarrollo del trabajo, comunicándose entre sí. Por otra parte, vamos a tener el nodo Robot, con el cuál nos comunicamos para recibir información y mandar órdenes de movimiento, aunque este último nodo está ya creado.

Nodo Movimiento

Este será el nodo principal de todo el programa, es el nodo que se encarga de comunicarse con el robot y con la interfaz, obteniendo parámetros y valores de cada uno de los nodos y realizando todos los cálculos que van a permitir que se consiga la funcionalidad deseada.

Este nodo va a tener el objetivo de calcular la posición y orientación del robot a partir de unos valores de incremento de la punta de la herramienta. Esta funcionalidad se ha descrito de forma teórica en el Capítulo 2, y en este apartado se va a mostrar la forma de programar el nodo para que se pueda desarrollar dicha función.

Al ejecutar este nodo, lo primero que se hace es esperar a recibir parámetros de configuración, sin los cuales no puede desarrollar el código, así como recibir la posición y orientación actuales del robot. Tras esto, se vuelve a poner a la espera de recibir algún comando de movimiento para la punta de la herramienta. Cuando recibe estos comandos, habilita una parte del código que se encontraba en espera hasta tener tanto los valores de configuración como los de movimiento. Esta parte se encarga de realizar todos los cálculos con los parámetros recibidos por cada topic, y de mandar la posición que debe adoptar el manipulador para que la punta de la herramienta alcance la posición deseada.

El nodo Movimiento va a estar dividido en dos partes:

Por un lado encontramos una parte dedicada a la configuración del nodo. Esta parte va a contener la importación de todas las librerías de las cuales se va a hacer uso, así como la declaración de variables importantes, y la programación de las funciones Callback() y Conversión de Euler a Matriz de rotación.

- **Librerías:** Algunas son solo necesarias para ejecutar el nodo en ROS, pero otras se encargan de realizar cálculos más complejos como las conversiones entre representaciones de la orientación. Las librerías que se utilizan en este programa son:
 - **Rospy:** Esta librería permite escribir el código del nodo de ROS en Python.

- **Math:** Contiene funciones de gran utilidad como realizar raíces cuadradas o funciones trigonométricas.
- **Numpy:** Permite realizar cálculos computacionales de forma más sencilla.
- **Std_msgs.msg.:** Permite declarar variables de diferentes tipos.
- **Pytransform3d:** Esta librería contiene las funciones que nos permitirán realizar las conversiones de una representación de la orientación a otra, de forma que no tengamos que realizar tantos cálculos.

```
#!/usr/bin/env python

import rospy
import math
import numpy as np
from std_msgs.msg import String
from std_msgs.msg import Float64MultiArray as FM
from pytransform3d.rotations import matrix_from_axis_angle
from pytransform3d.rotations import axis_angle_from_matrix
from pytransform3d.rotations import matrix_from_euler_zyx
```

- **Declaración de variables:** Tan solo se crea la variable, con valor 0, para poder asignar más tarde el valor que se desee. Algunas de estas variables son globales, lo que nos permite usarlas en diferentes funciones del programa, mientras que otras son solo utilizadas en una sola función. Los tipos de variable que se han utilizado en este código son:
 - **String:** Este tipo permite declarar variables como una cadena de caracteres alfanuméricos.
 - **Float64MultiArray:** Este tipo de variable pertenece a ROS, y consiste en un tipo de variable con seis parámetros, que en nuestro caso harán referencia a los tres valores de posición referentes a cada eje (x, y, z), y a los tres valores de orientación del tipo vector de rotación.
 - **Float:** Este tipo sirve para declarar variables de números decimales.
 - **Int:** Permite crear variables de números enteros.
 - **Bool:** Son variables del tipo Verdadero o Falso, que nos servirá precisamente para habilitar la segunda parte del nodo.

- **Funciones Callback()**: Estas funciones son llamadas cada vez que se recibe un mensaje por un topic. Para cada topic se ha creado una función Callback():

- **Callback1:** Esta función será llamada cada vez que se recibe un nuevo mensaje por el topic “UR3_1/outputs/pose”. Lo que se hace en esta función es obtener la pose suministrada por el robot, y guardar en una variable distinta cada uno de los valores de posición y orientación.

```
def callback1(data):
    global xi, yi, zi, ox, oy, oz

    #Recibimos la posicion y orientacion actual del robot
    xi = data.data[0]
    yi = data.data[1]
    zi = data.data[2]
    ox = data.data[3]
    oy = data.data[4]
    oz = data.data[5]

    rate = rospy.Rate(10)
    #rate.sleep()
```

- **Callback2:** Esta función se llama cuando llega un nuevo mensaje por el topic “configuracion”. Se encargará de obtener los valores de longitud de la herramienta y de posición relativa del punto de fulcro y guardarlas en variables.

```
def callback2(data):
    global vector, IP, fulcro, orientacion, longitud

    #Recibimos valores de configuracion de la interfaz y los sepamos
    vector = data.data.split()

    fulcro = float(vector[0])
    longitud = float(vector[1])
```

- **Callback3:** Se llama cuando se reciben mensajes por el topic “movimiento”. Va a guardar los valores de incremento de posición de la punta de la herramienta en variables que se usarán más tarde, y va a dar la señal que permita iniciar la segunda parte del programa.

```

def callback3(data):
    global Ph1, Ph2, Ph3, iteracion

    #Recibimos incrementos para mover la pinza
    Ph1 = data.data[0]
    Ph2 = data.data[1]
    Ph3 = data.data[2]

```

- **Función Euler ZYZ:** Esta no es una función Callback(), pero es una función a la que se va a llamar cada vez que se quiera realizar la conversión de ángulos de Euler a Matriz de rotación. Contiene los cálculos descritos en el apartado de conversiones del Capítulo 2.

```

def eulerXYZ(v):
    global M

    #Esta funcion va a crear la matriz de orientacion para nuestro robot

    M = [[math.cos(v[0])*math.cos(v[1])*math.cos(v[2]) -
math.sin(v[0])*math.sin(v[2]), -
math.cos(v[0])*math.cos(v[1])*math.sin(v[2]) -
math.sin(v[0])*math.cos(v[2]), math.cos(v[0])*math.sin(v[1])],
       [math.sin(v[0])*math.cos(v[1])*math.cos(v[2]) +
math.cos(v[0])*math.sin(v[2]), -
math.sin(v[0])*math.cos(v[1])*math.sin(v[2]) +
math.cos(v[0])*math.cos(v[2]), math.sin(v[0])*math.sin(v[1])],
       [-math.sin(v[1])*math.cos(v[2]), math.sin(v[1])*math.sin(v[2]),
math.cos(v[1])]]

    return M
    #La variable iteracion nos servira para mover el robot en caso de que sea
True
iteracion = True

```

En la segunda parte del código se encontrarán todos los cálculos para conseguir la posición y orientación deseadas. Va a contener la inicialización del nodo, y el bucle que realizará los cálculos al recibir los comandos de movimiento:

- **Inicialización del nodo:** Contiene la configuración del nodo, así como las suscripciones y las publicaciones a cada topic.
- **Bucle de cálculos:** Como su nombre indica, se trata de una parte del código que se repetirá de forma indefinida hasta que se cierre la aplicación. Este bucle se encuentra a la espera hasta que recibe los comandos de movimiento, entonces comienza a realizar las operaciones mencionadas en el Capítulo 2 hasta

conseguir la pose y publicarla, tras ello se vuelve a poner a la espera de recibir nuevos comandos de movimiento.

Nodo Interfaz

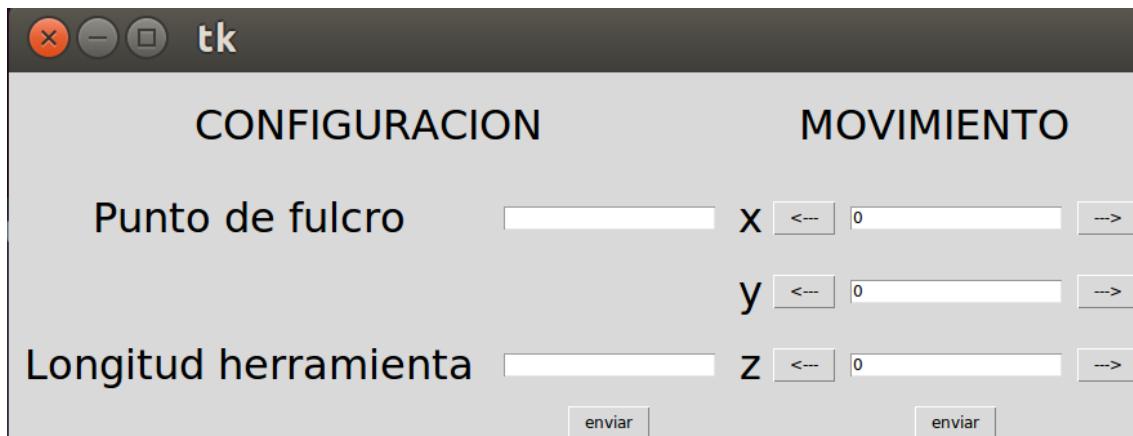


Figura 14. Apariencia de la interfaz.

Este nodo va a ser el encargado de declarar algunas variables y de publicar los movimientos que se desean realizar sobre la punta de la herramienta.

Al igual que el nodo movimiento, se divide en dos partes.

Una primera parte en la que se va a encontrar la importación de las librerías necesarias y la declaración de variables.

- **Librerías:** Al igual que en el nodo de Movimiento, tendremos algunas librerías necesarias para el correcto funcionamiento y algunas para poder generar el código.
 - **Rospy:** Como se menciona anteriormente, es necesaria para generar el código en el lenguaje de Python.
 - **Std_msgs.msgs:** Para la declaración de variables.
 - **Tkinter:** Esta es una librería destinada a la generación del interfaz. Contiene funciones que facilitarán la creación de los botones, etiquetas, y cuadros de texto.

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
from std_msgs.msg import Float64MultiArray as FM
from Tkinter import *
root = Tk()
```

- **Declaración de variables:** Se utilizarán variables del tipo **String**, **Float**, **Int** y **Float64MultiArray**.

```
fulcro = StringVar()
longitud = StringVar()
ejex = StringVar()
ejey = StringVar()
ejez = StringVar()
x = 0
y = 0
z = 0
```

Y una segunda parte que va a realizar la configuración de la interfaz, la cuál nos permitirá introducir los valores y parámetros que queremos publicar para nuestro robot. Dentro de esta parte, se ha hecho uso de las funciones que nos proporciona la librería **Tkinter**, que son:

- **Label:** Esta función está destinada a la creación de etiquetas de texto, que nos servirán para indicar qué vamos a introducir en cada cuadro de texto. Se puede configurar su tamaño y posición dentro de la interfaz.
- **Entry:** Esta función se encarga de generar la entrada de texto dentro de un cuadro, que será donde introduciremos los valores de posición de punto de fulcro relativa y longitud de la herramienta, así como los valores de incremento de posición de la punta de la herramienta.
- **Button:** Encargada de crear un botón que realizará una función específica al pulsar, en nuestro caso será llamar a una función que publicará los valores introducidos para que el nodo Movimiento pueda recibirlas.

El código está distribuido de forma que primero se genere la parte de configuración, la cuál se encontrará a la izquierda en la interfaz, como puede verse en la Figura 14, y en la cuál vamos a tener primero las funciones que crean las entradas de texto con sus etiquetas, y el botón de enviar. Al pulsar el botón enviar, se llamará a la función **variables()**, que es una función creada en la cuál se guardan las dos variables de punto de fulcro y longitud de la herramienta dentro de la variable configuración, y se publica por el topic de “configuración”.

Por otro lado tendremos la parte de movimiento, que estará distribuido en tres partes referentes a los tres ejes X, Y, Z. En cada una de ellas, se encuentra la configuración de las etiquetas y de la entrada del cuadro de texto, así como dos botones, uno que suma y otro que resta, por tanto, tendremos dos botones para cada eje más el botón de enviar que tiene la misma función que el botón de enviar de la otra parte de la interfaz.

Al pulsar los botones de sumar o restar, lo que se hace es llamar a una función que sumará o restará un incremento de 0.01 metros dependiendo del eje al que se refiere el botón. Por último encontramos el botón enviar, que recoge en una variable movimiento de tipo `Float64MultiArray`, los valores de los tres ejes juntos para publicarlo por el topic “movimiento”.

Como se puede ver en la Figura 14, se puede escribir directamente el incremento para cada eje escribiéndolo, o sumar y restar pequeños incrementos a través de los botones laterales.

3.3 URXdriver

URXdriver es un paquete de ROS desarrollado por el grupo de robótica médica de la Universidad de Málaga que nos permitirá controlar el movimiento del robot, así como una visualización 3D para aplicaciones de ROS. Representa el modelo de robot que queremos simular, y es capaz de obtener datos de cámaras, sensores, etc. En nuestro caso, será quien comunique nuestro nodo con el simulador de Universal Robots, por tanto, tendremos que conectarlo a la simulación de nuestra máquina virtual a través de los archivos de configuración. Se encargará de mostrar los movimientos de simulación que nos permitirá conocer los movimientos que irá realizando nuestro manipulador a medida que se ejecuta el código.

Para configurarlo tendremos que modificar el archivo de configuración, en el que podremos cambiar la dirección IP a la que se debe conectar y poner la IP del robot.

La apariencia que tiene esta herramienta se puede ver en la Figura 15.

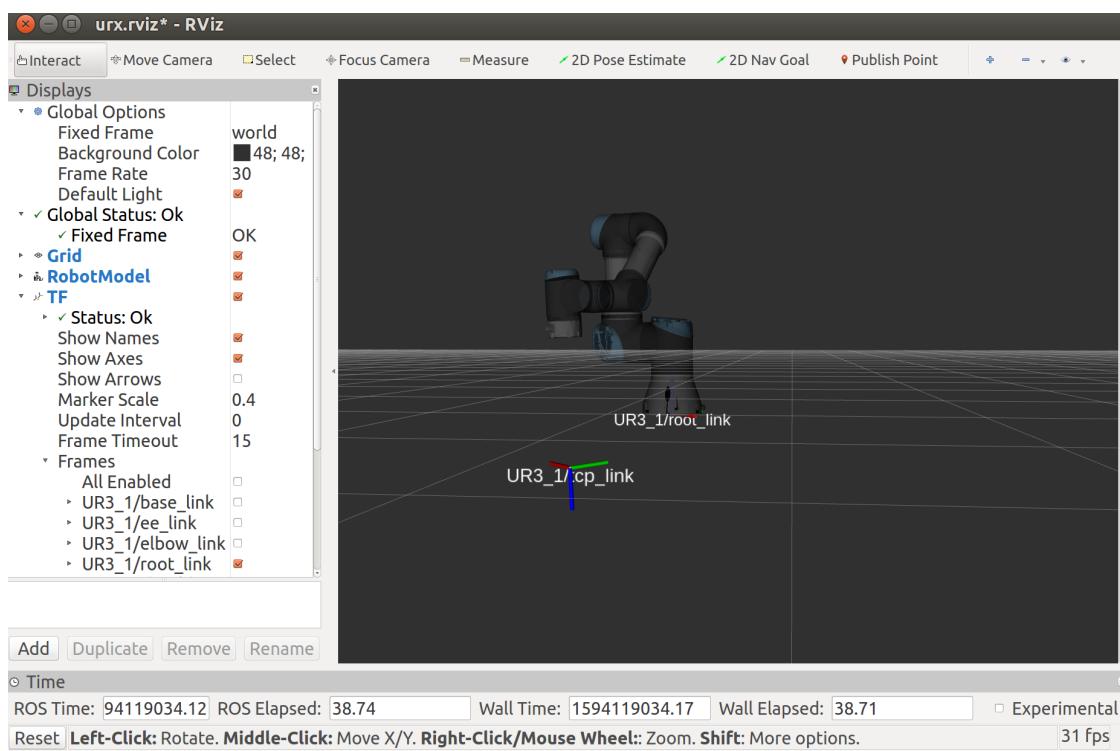


Figura 15. Simulador RViz representando el robot UR3.

3.4 URSim

URSim es una herramienta de simulación como RViz, pero desarrollada por la empresa de Universal Robots. Este software debe instalarse sobre una máquina virtual, y contiene tres aplicaciones, referentes a los modelos de sus robots, con los que se pueden crear programas para simularlos en ellos. En mi caso se utiliza el simulador de UR3, con el que se puede colocar al robot en cualquier posición y orientación. Este será el robot virtual que realizará el movimiento, mientras que la herramienta de URXdriver, a través de la conexión de IP, lo que hace es imitar los movimientos de este robot. Al imitar los movimientos, como es una herramienta de ROS, podemos obtener todos estos movimientos mediante topics.

3.5 UR3 de Universal Robots

El robot para el cuál se va a realizar este proyecto es el UR3 (Figura 16), de Universal Robots. Esta empresa se fundó en el año 2005 por Esben Østergaard, Kasper Støy y Kristian Kassow con el objetivo de conseguir que la robótica fuera un tipo de tecnología accesible para pequeñas y medianas empresas. En 2008 comercializan su primer robot, el UR5, considerado uno de los logros tecnológicos más significativos de la década en el ámbito robótico. Más tarde en el 2012 lanzan el UR10, diseñado para tareas más complejas y de mayor capacidad de fuerza.



Figura 16. Robot UR3 de Universal Robots.

4 EXPERIMENTACIÓN Y RESULTADOS

En este capítulo se van a mostrar los resultados obtenidos tras realizar las pruebas necesarias que nos permitirán conocer la validez del trabajo desarrollado, dejando ver si los objetivos propuestos se alcanzan correctamente. Para empezar, se realizaron experimentos previos al desarrollo del trabajo que permitieron obtener un aprendizaje sobre las herramientas de simulación y sobre la comunicación con el robot. Todos los códigos desarrollados para la realización de este trabajo, así como el vídeo en el que se muestra el funcionamiento de todos los nodos en conjunto para conseguir el objetivo están disponibles en el siguiente enlace: https://github.com/irivas-uma/TFG_PabloFuentes

Pruebas iniciales

Estas primeras pruebas consisten en colocar el robot de simulación en una posición y una orientación conocidas y obtener su pose a través de nuestro nodo de movimiento. Esto nos permite saber si la obtención de valores a través de los topics es correcta. Para el primer caso se colocará el robot en una posición cualquiera, pero orientado mirando hacia abajo, es decir, con el vector dirección del efector final como $(0, 0, -1)$. Esta prueba se realiza debido a que el robot nos proporciona su orientación en formato vector de rotación, y se quiere comprobar si al realizar las conversiones necesarias se obtiene que la dirección del efector final es la indicada anteriormente.

En la simulación colocamos el robot mirando hacia abajo, como se ve en la Figura 17:



Figura 17. Simulador de URSim con la configuración inicial.

Una vez colocado el robot, lanzamos el simulador URXdriver, que nos permite comunicarnos con el robot simulado. Con la conexión establecida, primero comprobamos los valores recibidos por el robot ejecutando en el terminal el comando **rostopic echo UR3_1/outputs/pose**, este comando nos permite mostrar en pantalla lo que se manda por el topic, en este caso es el topic por el que se publica la pose del robot.

Al mismo tiempo vamos a lanzar el nodo en el que se realiza la conversión de la orientación, de forma que podremos comprobar si realmente se calcula como es debido.

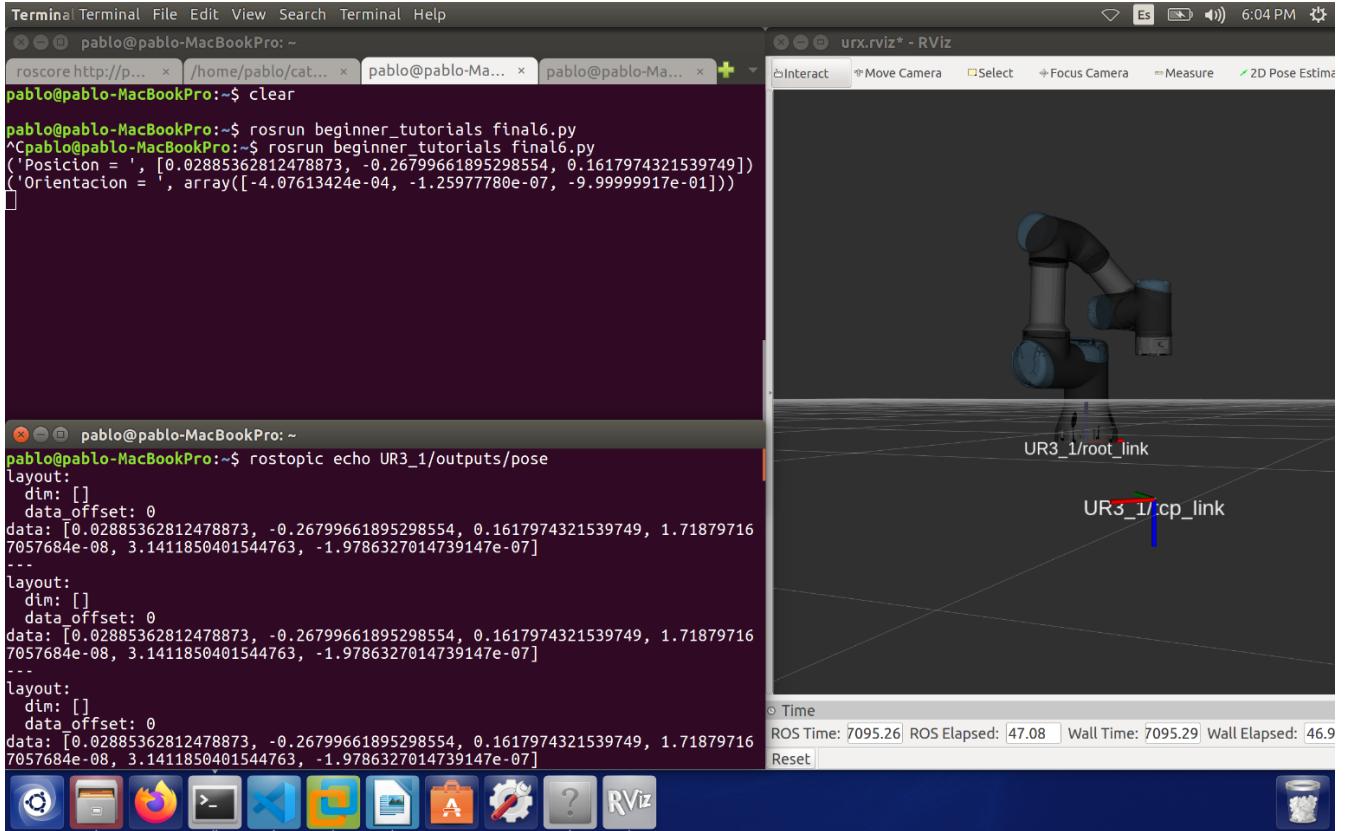


Figura 18. Obtención de pose del robot.

En la Figura 18 puede verse en el lado derecho el simulador que se comunica con el robot virtual. En la ventana de terminal de abajo se muestra lo recibido por el topic, que se puede ver cómo los tres primeros valores hacen referencia a la posición y los tres últimos a la orientación, que coincide con la figura 18, aunque en la terminal se muestran en metros y en el simulador URSim se muestran en milímetros. Por último, en la ventana superior del terminal se ejecuta el nodo que transforma la orientación. Se ve cómo la posición coincide, pero la orientación ya se muestra transformada en vector de dirección, y como anteriormente se comenta, este vector sale como (0, 0, -1) aproximadamente.

Con esto sabemos que recibimos correctamente la pose, y la conversión necesaria de la orientación se calcula sin errores. Por tanto, se procede a realizar los experimentos que nos permiten saber si se ha logrado calcular la posición y orientación

del robot para los incrementos de posición de la herramienta deseados. Para ello, vamos a obtener la posición final del efecto final del robot y de la punta de la herramienta, y uniendo ambos puntos se obtiene una representación de la herramienta. Al hacerlo con distintos incrementos, se consigue representar varias veces la herramienta, pasando todas las veces por el punto de fulcro.

Para realizar esto, además de lo utilizado en el experimento anterior, se va a hacer uso de la herramienta Matlab. Con este programa podremos dibujar la línea que une los puntos del efecto final y de la punta de la herramienta, para poder representar la herramienta en cada incremento.

Se van a realizar las pruebas para diferentes tamaños de herramienta, y diferentes posiciones relativas del punto de fulcro al efecto final, que son los dos parámetros de configuración inicial que se proporcionan por la interfaz.

Experimento 1

Se va a establecer como longitud de herramienta 400 milímetros y como distancia relativa al punto de fulcro 200 milímetros, de forma que el punto de fulcro se encuentra entre el efecto final y la punta de la herramienta. Estos valores se introducen en metros.

Esta primera prueba la vamos a realizar en una sola dimensión, es decir, le vamos a incrementar solo la posición en uno de los ejes, el Eje X.

Establecemos los valores como en la Figura 19:

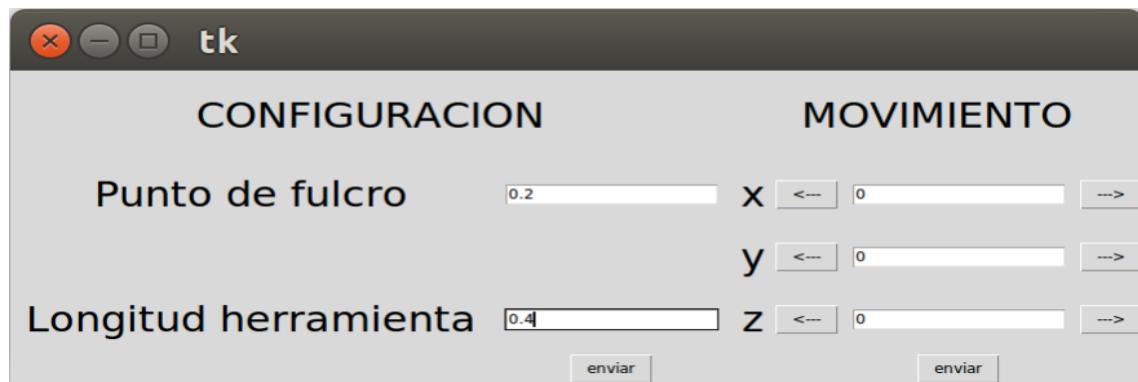


Figura 19. Determinación de los valores de configuración.

Realizamos una serie de movimientos:

```
roscore http://pablo... x pablo@pablo-MacBo... x pablo@pablo-MacBo... x pablo@pablo-MacBo... x + 
NUEVO MOVIMIENTO
('Incremento', [0.03, 0.0, 0.0])
('Nueva posicion efector Pn', array([-0.00081665, -0.26792858, 0.15907218]))
('Nueva posicion herramienta Ptn', [0.0472225748086689, -0.2580194222873031, -0.237909018519
4403])
Comprobaciones
('Actual posicion herramienta Pt', array([ 0.01722226, -0.25801942, -0.23790902]))
('Longitud herramienta Dt', 0.4)

NUEVO MOVIMIENTO
('Incremento', [-0.05, 0.0, 0.0])
('Nueva posicion efector Pn', array([ 0.04844757, -0.26791815, 0.15867436]))
('Nueva posicion herramienta Ptn', [-0.027750866113425005, -0.25801992872887314, -0.23790887
26782757])
Comprobaciones
('Actual posicion herramienta Pt', array([ 0.04722491, -0.25801993, -0.23790887]))
('Longitud herramienta Dt', 0.4000000000000001)

NUEVO MOVIMIENTO
('Incremento', [0.02, 0.0, 0.0])
('Nueva posicion efector Pn', array([ 0.02884905, -0.26799571, 0.16179717]))
('Nueva posicion herramienta Ptn', [0.01722676462669841, -0.25802030039254736, -0.23790949247
45595])
Comprobaciones
('Actual posicion herramienta Pt', array([-0.00277324, -0.2580203, -0.23790949]))
('Longitud herramienta Dt', 0.4)
```

Figura 20. Obtención de resultados en el terminal.

```
NUEVO MOVIMIENTO
('Incremento', [0.07, 0.0, 0.0])
('Nueva posicion efector Pn', array([-0.03505633, -0.26751798, 0.14282207]))
('Nueva posicion herramienta Ptn', [0.08722684852819274, -0.2580249024774856, -0.237909728698
83553])
Comprobaciones
('Actual posicion herramienta Pt', array([ 0.01722685, -0.2580249, -0.23790973]))
('Longitud herramienta Dt', 0.4000000000000001)

NUEVO MOVIMIENTO
('Incremento', [-0.02, 0.0, 0.0])
('Nueva posicion efector Pn', array([-0.0191046, -0.26776045, 0.15254137]))
('Nueva posicion herramienta Ptn', [0.06722714831506912, -0.25802475844806294, -0.23790970554
445662])
Comprobaciones
('Actual posicion herramienta Pt', array([ 0.08722715, -0.25802476, -0.23790971]))
('Longitud herramienta Dt', 0.4)
```

Figura 51. Obtención de resultados en el terminal.

Lo que se observa en la terminal de las figuras 20 y 21 en cada nuevo movimiento es:

1. La nueva posición de nuestro efecto final al realizar los cálculos.
2. La nueva posición que va a tener nuestra punta de la herramienta (al añadirle los incrementos a la actual posición).

Por otra parte tendremos las comprobaciones. Estas comprobaciones son cálculos que se realizan dentro del nodo y que según el valor obtenido sabremos si se está realizando bien la operación.

1. La comprobación de la actual posición de la herramienta, que nos dice si la orientación proporcionada al robot es correcta, ya que estamos calculando la posición de la punta a través de esta orientación, y si fuera errónea no coincidiría con la posición de la punta de la herramienta del movimiento anterior. Para un mejor entendimiento véase la figura 21. En esta imagen vemos dos movimientos realizados, en el que el segundo movimiento, en el apartado de comprobaciones, la actual posición de la herramienta coincide con la nueva calculada en el anterior movimiento.
2. Longitud de la herramienta. Se calcula la distancia que hay entre la posición del efecto final y de la punta de la herramienta, haciendo el módulo del vector que los une, si coincide con la longitud establecida en la interfaz, se estará realizando un cálculo de la posición del efecto final correcto.

Para comprobar ahora los movimientos realizados de forma gráfica, se hace uso de Matlab, para representar los puntos finales del efecto y de la punta de la herramienta unidos, que hacen una representación de la herramienta:

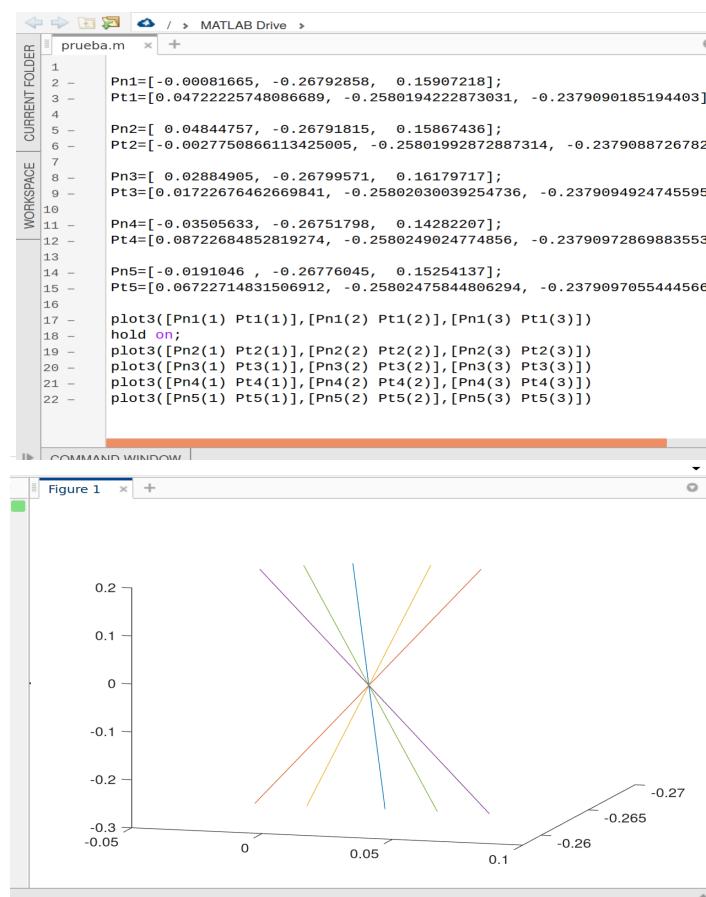


Figura 22. Representación de las herramientas en Matlab.

En la Figura 22 se observa cómo cada una de las herramientas pasa por un mismo punto, el punto de fulcro.

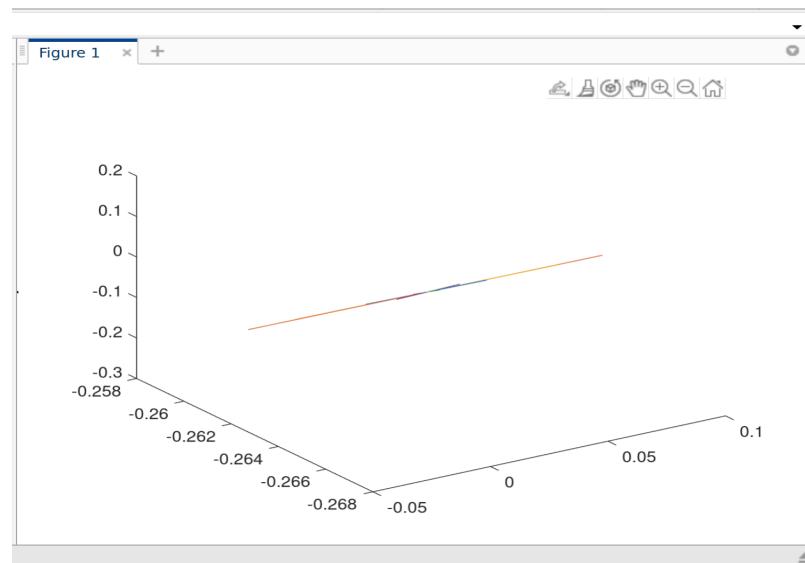


Figura 63. Gráfica mostrada desde arriba.

En esta figura se ha colocado la gráfica de forma paralela a las herramientas, para poder comprobar que el movimiento se realiza solo en un eje, como se quería realizar el experimento.

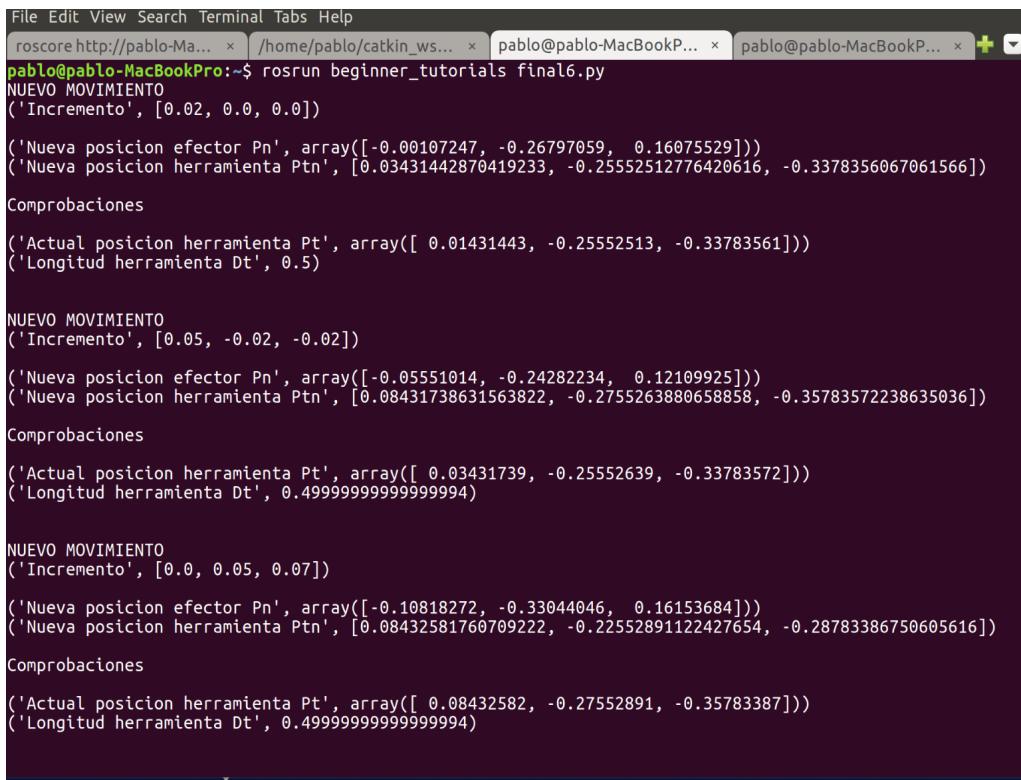
Experimento 2

En esta segunda prueba realizaremos el experimento en tres dimensiones, para ver que el funcionamiento es correcto con un caso más real, ya que en una cirugía, la herramienta se moverá en las tres dimensiones. Además, se asignarán diferentes valores de longitud de la herramienta y punto de fulcro relativo para hacer la comprobación de que el programa funciona con cualquier valor de esos parámetros.

Estableceremos los parámetros de longitud de la herramienta como 500 milímetros, y la distancia relativa del fulcro a 300 milímetros del efecto final, introducidos en metros, como se muestra en la Figura 24.

Figura 74. Determinación de los valores de configuración.

A continuación se realizan una serie de movimientos con la herramienta, con la que veremos los resultados en la terminal de las figuras 25 y 26.



```

File Edit View Search Terminal Tabs Help
roscore http://pablo-Ma... /home/pablo/catkin_ws... pablo@pablo-MacBookP... pablo@pablo-MacBookP...
pablo@pablo-MacBookPro:~$ rosrun beginner_tutorials final6.py
NUEVO MOVIMIENTO
('Incremento', [0.02, 0.0, 0.0])

('Nueva posicion efector Pn', array([-0.00107247, -0.26797059, 0.16075529]))
('Nueva posicion herramienta Ptn', [0.03431442870419233, -0.25552512776420616, -0.3378356067061566])

Comprobaciones

('Actual posicion herramienta Pt', array([ 0.01431443, -0.25552513, -0.33783561]))
('Longitud herramienta Dt', 0.5)

NUEVO MOVIMIENTO
('Incremento', [0.05, -0.02, -0.02])

('Nueva posicion efector Pn', array([-0.05551014, -0.24282234, 0.12109925]))
('Nueva posicion herramienta Ptn', [0.08431738631563822, -0.2755263880658858, -0.35783572238635036])

Comprobaciones

('Actual posicion herramienta Pt', array([ 0.03431739, -0.25552639, -0.33783572]))
('Longitud herramienta Dt', 0.4999999999999994)

NUEVO MOVIMIENTO
('Incremento', [0.0, 0.05, 0.07])

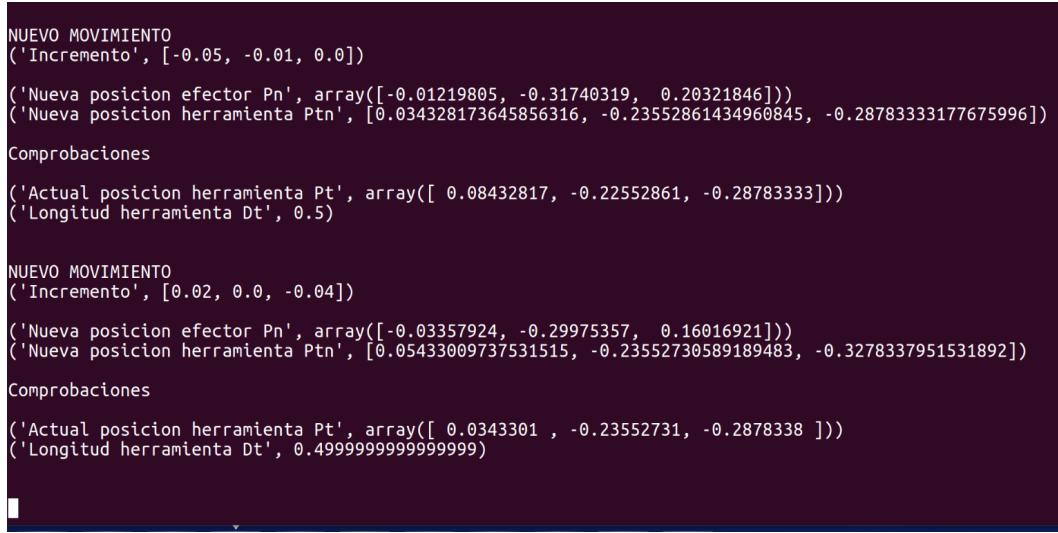
('Nueva posicion efector Pn', array([-0.10818272, -0.33044046, 0.16153684]))
('Nueva posicion herramienta Ptn', [0.08432581760709222, -0.22552891122427654, -0.28783386750605616])

Comprobaciones

('Actual posicion herramienta Pt', array([ 0.08432582, -0.27552891, -0.35783387]))
('Longitud herramienta Dt', 0.4999999999999994)

```

Figura 85. Obtención de resultados en terminal.



```

NUEVO MOVIMIENTO
('Incremento', [-0.05, -0.01, 0.0])

('Nueva posicion efector Pn', array([-0.01219805, -0.31740319, 0.20321846]))
('Nueva posicion herramienta Ptn', [0.034328173645856316, -0.23552861434960845, -0.2878333177675996])

Comprobaciones

('Actual posicion herramienta Pt', array([ 0.08432817, -0.22552861, -0.28783333]))
('Longitud herramienta Dt', 0.5)

NUEVO MOVIMIENTO
('Incremento', [0.02, 0.0, -0.04])

('Nueva posicion efector Pn', array([-0.03357924, -0.29975357, 0.16016921]))
('Nueva posicion herramienta Ptn', [0.05433009737531515, -0.23552730589189483, -0.3278337951531892])

Comprobaciones

('Actual posicion herramienta Pt', array([ 0.0343301, -0.23552731, -0.2878338 ]))
('Longitud herramienta Dt', 0.499999999999999)

```

Figura 96. Obtención de resultados en terminal.

Con estos resultados se puede realizar las comprobaciones anteriores, con las que podemos ver que en cada iteración la actual posición de la herramienta coincide con el valor de la iteración anterior de nueva posición de la herramienta, lo que indica que la orientación es correcta. Además se observa cómo el valor de longitud de herramienta

calculado a partir de los valores finales de posición del efecto final y de la herramienta coincide con el valor establecido en la interfaz.

Al llevar estos valores a una representación en Matlab se obtiene lo mostrado en la Figura 27:

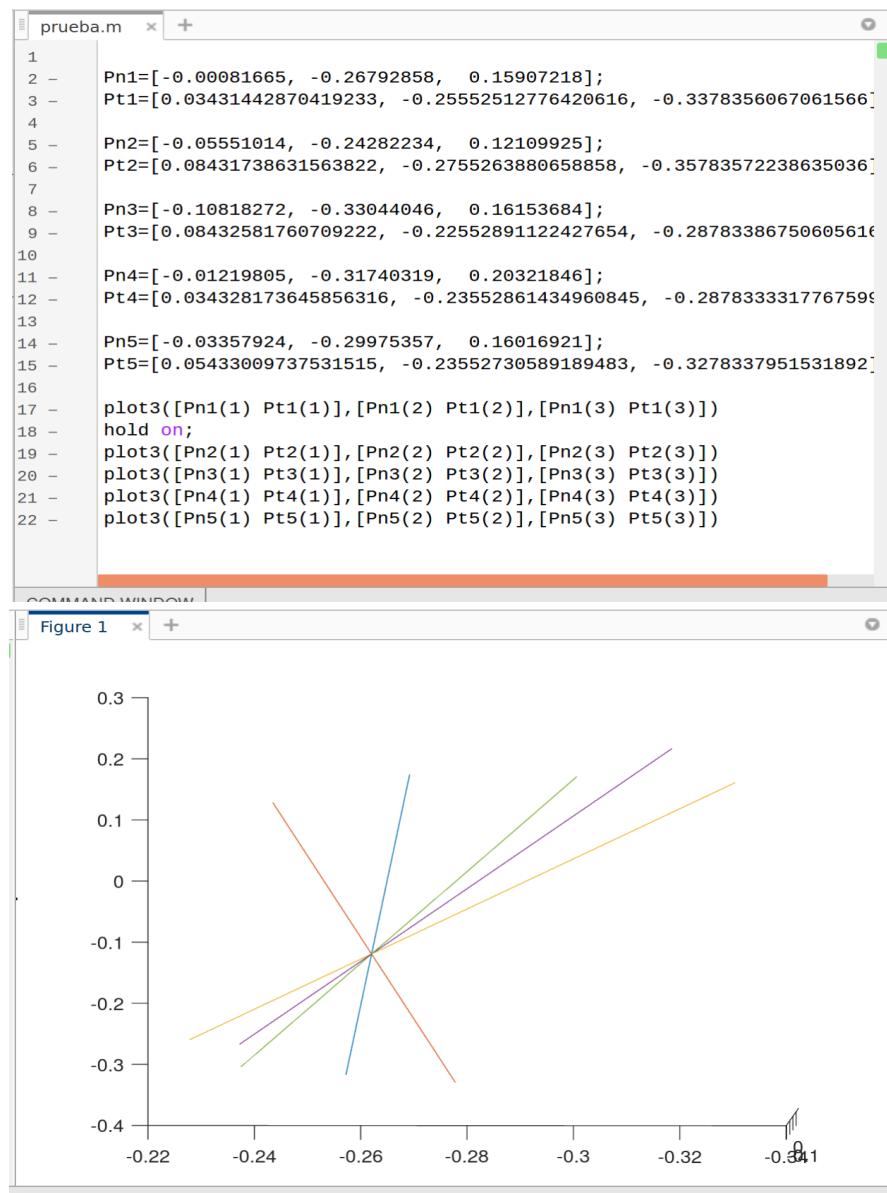


Figura 27. Representación de las herramientas en Matlab.

Como se observa en esta primera imagen, pasan todas las rectas por el mismo punto, que hace referencia al punto de fulcro.

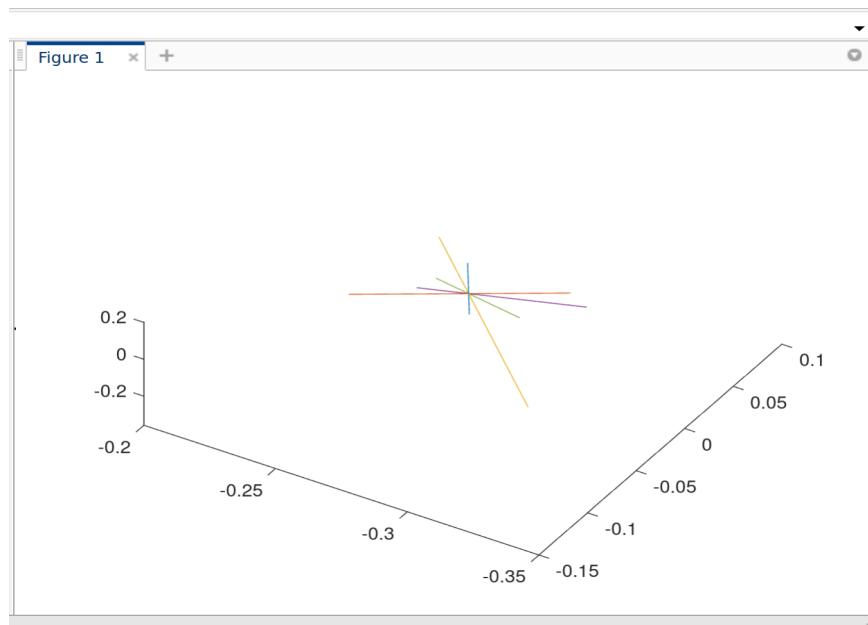


Figura 28. Gráfica mostrada desde arriba.

Viendo la gráfica desde un punto de vista más superior en la Figura 28, se observa cómo ya no se realiza todo el movimiento en una sola dirección, sino que se mueve por todo el espacio.

Para comprobar si el punto por el que se cruzan todas las rectas coincide con el punto de fulcro calculado, se ha extraído dicho valor del programa para los experimentos con longitud de herramienta de 500 milímetros y posición relativa del fulcro en 300 milímetros, y el valor es [0.02013009, -0.26051372, -0.13798243]. Se ha colocado la gráfica en paralelo a los tres ejes para comprobar que el punto coincide en este valor.

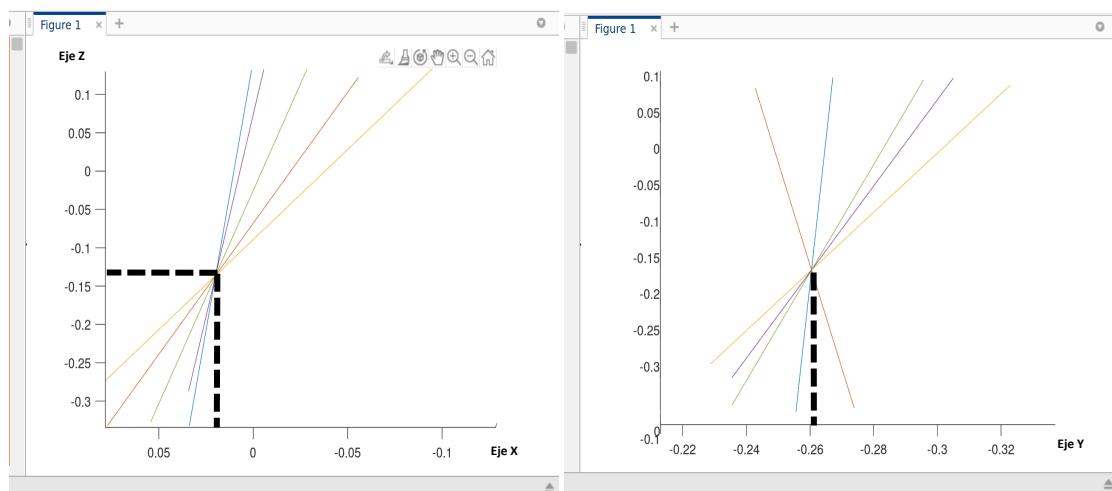


Figura 29. Demostración de la correcta posición del punto de fulcro.

En la izquierda de esta Figura se muestran los ejes X en horizontal (0.02) y el Z en vertical (-0.13), que coinciden con el valor extraído del nodo. A la derecha se muestra el eje Y en horizontal (-0.26), que también coincide.

5 CONCLUSIONES Y FUTUROS TRABAJOS

En este documento se ha presentado una aplicación robótica que permite realizar el control de posición y orientación de un manipulador con una herramienta quirúrgica de forma que puede alcanzar cualquier posición con la punta de dicha herramienta dentro del cuerpo de un paciente, limitada por un punto de fulcro. Esta aplicación se ha desarrollado en lenguaje Python, sobre el sistema operativo de ROS, lo que hace que sea una aplicación genérica y se pueda ejecutar en cualquier robot que permita instalar este software, a pesar de que se ha desarrollado utilizando solo el robot UR3 de Universal Robots. Se ha implementado además una interfaz gráfica diseñada en Python que ayuda a la declaración de parámetros variables según el manipulador sobre el cuál se va a ejecutar el programa, como es la longitud de la herramienta, y la posición relativa del punto de fulcro con respecto al efecto final del robot al empezar la operación. Esta interfaz contiene además un apartado de control del movimiento que permite indicar el incremento que se desea realizar con la herramienta. Con el apartado de experimentación y resultados se demuestra que se ha conseguido el objetivo de este trabajo.

A pesar de que se haya conseguido el principal objetivo, a esta aplicación se le pueden añadir una serie de mejoras o trabajos futuros, que pueden ser los siguientes:

- **Uso de la herramienta EndoWrist de Da Vinci:** En un principio, este trabajo está desarrollado sin tener en cuenta el tipo de herramienta que se utiliza, sin embargo, se desea que la herramienta que se vaya a usar para llevar a cabo esta aplicación es la herramienta EndoWrist del sistema Da Vinci. Se trata de una pinza con 7 grados de libertad que proporciona al cirujano multitud de posibilidades al poder realizar cualquier movimiento con la herramienta una vez el robot se ha posicionado y orientado en la pose deseada (Figura 30).

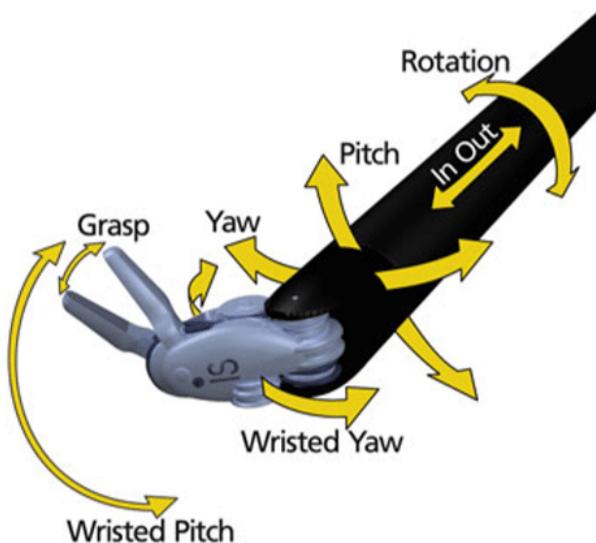


Figura 30. Grados de libertad del EndoWrist

- **Control de la herramienta a través de un dispositivo háptico:** Se puede sustituir el control de movimiento actual, desarrollado en una interfaz gráfica por un dispositivo háptico **Phantom Omni**. Habría que programar el Phantom para traducir los movimientos realizados con este en un incremento que se desea que realice la herramienta. Además se podría programar de forma que también pueda controlar los grados de libertad de la herramienta EndoWrist, y de esta forma se podrían hacer los controles tanto de movimiento como de acciones de la herramienta con un solo controlador.
- **Gráfica en tiempo real:** Para no tener que copiar los resultados obtenidos en la terminal del ordenador y pegarlos en Matlab cada vez que se realiza una operación, se podría programar una aplicación en este software que permitiera obtener los resultados en tiempo real desde ROS para que se fueran graficando a medida que se realiza un nuevo movimiento.

6 BIBLIOGRAFÍA

- [1] Jorge Gerardo Pereira Fraga. "*Present of the robotic surgery*".
- [2] Alfredo Córdova Dupeyrat, Garth H. Ballantyne. "*Sistemas Quiúrgicos Robóticos y Telerobóticos para cirugía abdominal*".
- [3] Carlos Martínez Ramos. "*Cirugía Robótica (I): Origen y evolución*".
- [4] A.H. Vilchis-González, J.C. Ávila-Vilchis, R.G. Estrada-Flores, R. Martínez-Méndez, O. Portillo-Rodríguez, M. Romero-Huertas. "*Modular Robots for Minimally Invasive Surgery*".
- [5] Andrés Jaramillo Botero, 2005. "*Descripciones y transformaciones espaciales*".
- [6] Carlo Tomasi. "*Vector Representation of Rotations*".
- [7] Rotation conversion for Robotics orientation -
<https://www.euclideanspace.com/maths/geometry/rotations/conversions/index.htm>
- [8] Robotic Operative System Homepage. <https://www.ros.org/>

